An Efficient Divide-and-Conquer Cascade for Nonlinear Object Detection

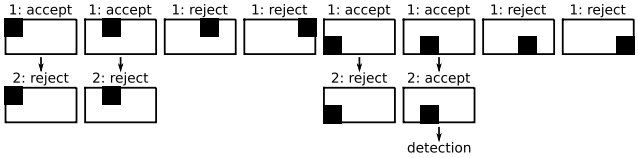
Christoph H. Lampert
Max Planck Institute for Biological Cybernetics, Tübingen, Germany
chl@tuebingen.mpg.de

Abstract

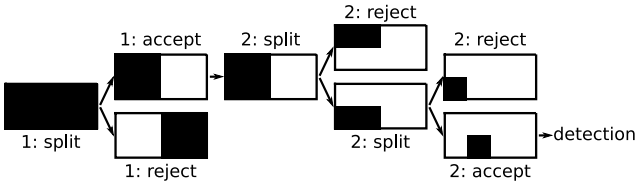
We introduce a method to accelerate the evaluation of object detection cascades with the help of a divide-and-conquer procedure in the space of candidate regions. Compared to the exhaustive procedure that thus far is the state-of-the-art for cascade evaluation, the proposed method requires fewer evaluations of the classifier functions, thereby speeding up the search. Furthermore, we show how the recently developed efficient subwindow search (ESS) procedure [11] can be integrated into the last stage of our method. This allows us to use our method to act not only as a faster procedure for cascade evaluation, but also as a tool to perform efficient branch-and-bound object detection with nonlinear quality functions, in particular kernelized support vector machines. Experiments on the PASCAL VOC 2006 dataset show an acceleration of more than 50% by our method compared to standard cascade evaluation.

1. Introduction

Reliable real-time object detection in natural images and videos is a long-time dream of computer vision. And indeed, in terms of their detection quality, general purpose object detectors have made significant progress over the last years. However, this is partially owed to the use of better classifier functions, more training images and multiple feature sets, all of which has increased the methods' runtime. Real-time systems for state-of-the-art object detection of arbitrary categories are therefore still not in reach. A particularly problematic class of methods are the most successful *sliding window* approaches, on which we will concentrate in this work. They detect objects by classifying a large number of candidate subwindows of the image whether they show an instance of the target object class or not. Because for natural images, locations, aspect ratio and size of the objects are all unknown, typically several hundred thousand regions have to be checked for each image, and each one requires a potentially costly classifier evaluation. To overcome this computational bottleneck, several techniques have been developed, most prominently *classifier cascades* and methods based on *global optimization*.



(a) Classical cascade evaluation: each possible candidate region is evaluated independently of the others (left to right). For each region, the stages are checked iteratively (top to bottom) until a negative classifier score occurs. If all scores are positive, the region is considered a detection.



(b) Efficient subwindow cascade: the algorithm starts with a single candidate set in stage 1 that contains all possible object locations (left). Depending on the quality bounds, the candidate region set is either accepted as a whole, rejected as a whole, or split into disjoint parts that are separately processed further. Accepted candidate sets are advanced to the next classifier stage or returned as detection if they already were in the last stage.

Figure 1. Schematics of cascade evaluation (simplified to 2D). Top: classical procedure; bottom: proposed divide-and-conquer method.

1.1. Classifier Cascades

Classifier cascades detect the location of object instances in an image by applying multiple classification functions sequentially to each possible candidate window [15, 20]. A candidate window is accepted as a detection only if all stages come to a positive decision. Each of the classifiers is fast, but relatively weak. The conjunction rule makes the set of weak individual classifiers into a single strong classifier, and it allows very fast evaluation of classifier cascades: as soon as one of the stages rejects a candidate region, we do not have to evaluate the later stages. Because most candidate regions in a sliding window setup do not show an object of interest, most cascade evaluations end after only few substages; the average number of classifier evaluations per window is much lower than the total length of the cascade.

One can further benefit from the fact that later stages in a cascade will be evaluated much less frequently than early

ones, by combining weak (and fast) classifiers in the early stages with strong (but slow) classifiers towards the end of the cascade. As result one obtain a more powerful classifier than from a cascade consisting only of weak classifiers, but which is still faster in object detection tasks than a single strong classifier, because the fast early stages filter out most negative candidates. Cascade classifiers of this type form the basis of many state-of-the-art object detection systems today, for example [4, 8, 19, 23].

1.2. Global Optimization

A second line of research for accelerating object detection tasks is the application of global optimization techniques. This is based on the observation that the exhaustive search pattern used by classical sliding window techniques is suboptimal—it ignores the fact that neighboring regions in image space typically have similar feature representations and therefore correlated classifier scores. Lampert *et al*. [10] introduced *efficient subwindow search (ESS)*, which performs a branch-and-bound search to identify the candidate region of highest classification score in an image. ESS is often more efficient than sliding-window approaches, because it uses information about potentially large sets of candidate regions in each step. More exactly, ESS bounds of the classification score over such region sets instead of computing the exact score of each region contained, and thereby ESS is able to avoid many computations for nonpromising image regions. Similar global optimization approaches have been proposed for nearest-neighbor based object detection [9, 21] and for action localization in videos [22].

Unfortunately, ESS achieves a significant acceleration only for simple – typically linear – classifiers. For nonlinear classifier functions, as they are necessary for object detection of highest-quality, one obtains only loose quality bounds, and a branch-and-bound search based on these does not yield a significant speedup. In [19], which describes one of the current best methods for object detection, Vedaldi *et al*. conclude about ESS: *"Unfortunately, we still found it necessary to visit several thousand regions per image, which [...] makes this approach inpractical. This motivates the use of a cascade of classifiers."*

In the rest of this paper, we will show that the use of cascaded classifiers and of global optimization techniques are not mutually exclusive concepts. We will introduce an evaluation scheme for classifier cascades, termed *efficient subwindow cascade (ESC)*, that combines the speed advantages of both techniques: it avoids classifier evaluations by relying on a representation of sets of regions instead of individual regions, and it supports the *early-stopping* property of classifiers that are the conjunction of simpler stages.

2. Efficient Subwindow Cascade

We first fix the notation for use in the following sections. Let f be a classifier function that assigns a score to any subregion of an image. Although the concept generalizes to arbitrary shapes, we only consider box-shaped regions here, *i.e*. $f : \mathcal{Y} \to \mathbb{R}$, where $\mathcal{Y}$ denotes the set of all rectangular subregions. To keep the notation concise, we do not write out the dependence of f the image we are currently considering. With $f(y) > 0$ indicating that a region y does show an instance of the object of interest, whereas $f(y) \leq 0$ means that it does not, *object detection* is defined as the task of computing

$$D := \{y \in Y : f(y) > 0\}, \tag{1}$$

i.e. the set of all regions with positive classification score.

Cascaded classifiers are a special case of the above situation, in which f is a conjunction of multiple *subclassifiers*, or *stages*, $f_1, \ldots, f_K$, such that for all $y \in \mathcal{Y}$

$$f(y) > 0 \;\Leftrightarrow\; \forall k \in \{1, \ldots, K\} : \; f_k(y) > 0. \tag{2}$$

When evaluating the subclassifiers iteratively, we obtain a *filtration*, $\mathcal{Y} = D_0 \supset D_1 \supset \cdots \supset D_K = D$, where the intermediate detection sets $D_k := \{y \in D_{k-1} : \; f_k(y) > 0\}$, approximate D with increasing quality.

2.1. Divide-and-Conquer Object Detection

The cascade structure provides us with a canonical way to determine D: for every $y \in \mathcal{Y}$, we iteratively test the subclassifiers. If $f_k(y) < 0$ for any $k < K$, we stop the evaluation because $y \notin D_k$ implies $y \notin D$. Figure 1(a) illustrates this process schematically for a two-stage cascade.

However, an exhaustive search over all elements $y \in \mathcal{Y}$ is not the only way to determine D and other methods might offer computational advantages. Schneiderman proposed *feature-centric* evaluation [16], but this requires a special form a additive classifier and did not find wide-spread use.

We will follow a *divide-and-conquer strategy* instead. Assume that functions $\underline{f}_k, \overline{f}_k : 2^{\mathcal{Y}} \to \mathbb{R}$ exist that bound the values of f_k from above and below, *i.e*. for any $Y \subset \mathcal{Y}$:

$$\underline{f}_k(Y) \leq f(y) \leq \overline{f}_k(Y), \qquad \text{for any } y \in Y, \tag{3}$$

and that the bounds are exact for single element sets,

$$\underline{f}_k(Y) = f(y) = \overline{f}_k(Y) \qquad \text{for } Y = \{y\}. \tag{4}$$

Then we formulate a new procedure for computing D that we call *efficient subwindow cascade (ESC)*. Algorithm 1 shows the algorithm in pseudo-code; Figure 1(b) illustrates it schematically. Its main difference between the new algorithm and an exhaustive search procedure is that ESC works with sets of candidate regions $Y \subset \mathcal{Y}$ as its basic working unit, instead of with individual regions $y \in \mathcal{Y}$.

Algorithm 1 Efficient Subwindow Cascade

Require: classifiers $f_1, \ldots, f_K$
Require: bounding functions $\underline{f}_1, \overline{f}_1, \ldots, \underline{f}_K, \overline{f}_K$
Output: D as defined by (1)

MAIN:

1: $D = \text{TRAVERSE}(\mathcal{Y}, 1)$
2: **return** D

TRAVERSE(Y, k):

3: **if** $\overline{f}_k(Y) \leq 0$ **then**
4: **return** $\emptyset$
5: **end if**
6: **if** $\underline{f}_k(Y) > 0$ **then**
7: **if** $k < K$ **then**
8: **return** TRAVERSE$(Y, k+1)$
9: **else**
10: **return** Y
11: **end if**
12: **end if**
13: split $Y_1 \dot{\cup} Y_2 \leftarrow Y$ with $Y_1 \neq \emptyset, Y_2 \neq \emptyset$
14: $D_1 = \text{TRAVERSE}(Y_1, k)$
15: $D_2 = \text{TRAVERSE}(Y_2, k)$
16: **return** $D_1 \cup D_2$

Starting with $Y = \mathcal{Y}$ in stage $k = 1$ *(line 1)*, ESC recursively traverses a binary search tree. In each step we first compute $\overline{f}_k(Y)$ *(line 3)*. If this is less than or equal to 0, we know that all elements $y \in Y$ have a negative score $f_k(y) \leq 0$, which implies $y \notin D$. Consequently we do not have to evaluate this subtree any further. Otherwise, we compute $\underline{f}_k(Y)$ *(line 6)*. If this is larger than 0, we know that all elements $y \in Y$ have a score $f_k(y) > 0$, which implies $y \in D_k$. Further processing of the elements of Y with f_k would not yield additional information. Therefore if $k < K$, we advance the whole subtree Y to the stage $k + 1$ *(line 8)*, otherwise, we return all elements of Y as detections *(line 10)*. If neither $\overline{f}_k(Y) \leq 0$ nor $\underline{f}_k(Y) > 0$ hold, the quality bounds were not yet tight enough to make a definite statement. We split the candidate set Y into two disjoint, non-empty parts *(line 13)* and we recursively process both subtrees. Note that such a split is always possible, because *line 13* can only be reached for Y that contain at least two elements. For single element sets, $Y = \{y\}$, condition (4) ensures that $\overline{f}_k(Y) = \underline{f}_k(Y)$, and consequently, either of the conditions in *line 3* or *line 6* would have been triggered. The correctness of the ESC algorithm follows from the following two propositions.

Proposition 1: Algorithm 1 terminates for any finite $\mathcal{Y}$.

Proposition 2: Algorithm 1 returns D as defined in Eq. (1).

The proofs are based on inductions over k and the size of Y. We omit them here for reasons of limited space.

2.2. ε-insensitive Detection

For practical purposes it is not always necessary to threshold the classifiers f_k exactly at 0. We can create an ε-*insensitive* version of Algorithm 1 by replacing the conditions in *line 3* by $\overline{f}_k(Y) \leq \varepsilon$, and in *line 6* by $\underline{f}_k(Y) > -\varepsilon$. The resulting algorithm will require fewer splits until convergence and therefore run faster, at the expense of returning only an approximate version of D. By similar a argument as above, one can show that all regions with at least one score $f_k(y) \leq -\varepsilon$ will be rejected, and that all regions with only scores $f_k(y) > \varepsilon$ will be accepted.

2.3. Integration of Efficient Subwindow Search

ESC relies on the same two concepts that are also at the core of ESS's efficiency: the interior representation that relies on *sets of regions* instead of individual regions and the *bounding functions* that can make statements about multiple object in a single step. The main difference between the two methods is the task they solve: ESC returns all regions of positive classification score, whereas ESS returns exactly one or some other predefined number. Because the number of output required from ESS is typically small, it benefits from a branch-and-bound strategy that targets its computation on only the few best states and prunes all others. This prevents ESS from being directly applicable to cascades; in the intermediate stages, statements about all candidate regions are required, not only about the ones of highest score. On the other hand, in the interior stages of a cascade, we also do not need the exact classification scores of the candidate regions, we only need their sign, and this is what enables ESC to accelerate cascade processing.

Note that for the last stage of the cascade the situation can be different. Depending on the intended use for the detection output, we might indeed only be interested in a few detections of the last stage, preferably the ones with maximal f_K score over D. Because the underlying representation of ESC and ESS are compatible, we can include an ESS-like branch-and-bound search procedure into ESC in this case: instead of running TRAVERSE in the K-th stage, we form a priority queue that contains all region sets that formed the output of stage $K - 1$ with priority value given by $\overline{f}_K$. From there we continue with the ESS branch-and-bound procedure. The initialization ensures that only regions $y \in D_{K-1}$ become candidates for ESS. Maximizing f_K over this set provides us with a list of elements of D_K (that is D) in order of decreasing f_K score. We stop the search after a predefined number of detections or when the scores become nonpositive, as this means that all of the elements of D have been identified.

Note that this trick works only in the last stage, whereas in the intermediate stages, regular ESC has to be used. This is because only ESC takes sets of regions as input as well as output. ESS makes use of region sets for initialization, but it

outputs individual regions. Consequently, if one tried, *e.g.*, to concatenate several ESS stages, all stage after the first would not benefit from the region set representation anymore and would work as ordinary sliding window stages.

2.4. Non-Maximum Suppression

When object detection is just one stage of a bigger system, one is often not really interested in all possible $y \in \mathcal{Y}$ of positive classification score, but only in a few representative locations that correspond best to the true object locations. All other elements of D correspond to duplicate detections of slightly different position or shape. To identify the relevant objects in D one typically applies *non-maximum suppression*, either in form of a clustering algorithm such as *mean-shift* [7], or by a greedy selection strategy that iteratively scans all detections in order of decreasing score and discards all regions that overlap too much with a previous, non-discarded detection.

Both setups are easy to apply to the output of ESC. For clustering approaches, we can make use of the fact that the output of ESC is already a representation of D in terms of region sets with adjacent image coordinates and presumably homogeneous image contents. We can therefore speed up the clustering by performing it on the level of region sets, *e.g.*, treating each of them as one sample in a mean-shift procedure, weighted by the number of regions it contains. Greedy techniques can be also be sped up, because the coordinate parameterization of the region sets allows us to estimate their minimal and maximal overlap with the detections so far. Consequently, we can discard many detections based only on their overlap, without the need to compute their exact score. This step is similar to adding an additional ESS-like stage to the ESC cascade and can be implemented using only the components already used in Algorithm 1.

3. Cascade Learning

The main topic of this paper is the problem of efficiently evaluating cascaded classifiers, not the question how to learn them from training data. However, ESC can also be used to speed up cascade training, as we will sketch in this section.

Detection cascades are typically trained by a bootstrapping procedure [20]: given a set of training images with annotation of the object locations, we train a first stage using the ground truth regions as positive training examples and randomly sampled background region as negative training examples. In subsequent stages, we extend the negative training sets by the false positive detection of the previous stage. In each stage, the classifier bias is adjusted such that as many examples as possible are rejected without introducing too many missed detections.

Iterative training of this kind is effective, but often slow. We can use ESC to mitigate this problem: because each initial section of the cascade is again a detection cascade, the step of identifying the false positives during training of a stage f_k can be performed efficiently by calling ESC to the stages $f_1, \ldots, f_{k-1}$.

4. Bounding Functions

In Section 2 we have assumed that quality bounding function $\overline{f}_k$ and $\underline{f}_k$ are given to us, which fulfill condition (3). In the following we will construct such bounds for different classifier functions, in particular for support vector machines (SVMs) with linear and commonly used nonlinear kernels. For simplicity of notation, we only consider histogram based image representation.

Upper bounds fulfilling same conditions as $\overline{f}_k$ are also required for ESS, and [10] derives them for linear SVMs. Similarly, [9] lists upper bounds for several distance functions in the context of nearest-neighbor classifiers, but these can also serve as components of SVM kernels. In contrast to previous methods, ESC requires not only upper bounds on the quality function, but also lower ones. These have not appeared in the literature before, but they can be derived using the same principle: in particular, let f be the decision function of a distance- or kernel-based classifier, *e.g.* a support vector machine:

$$f(y) = \sum\nolimits_i \alpha_i k(h^i, h^y) + b, \tag{5}$$

where h^y is a histogram representation of the image contents in the region y, the histograms $h^1, \ldots, h^m$ are prototypes, *e.g.* support vectors, $\alpha_1, \ldots, \alpha_m$ are real-valued coefficients, and b is a constant bias term. For any non-negative kernel or distance function[1] upper and lower bounds for any $Y \subset \mathcal{Y}$ are given by

$$\overline{f}(Y) = \sum_{\{\alpha_i > 0\}} \alpha_i \overline{k(h^i, h^Y)} + \sum_{\{\alpha_i < 0\}} \alpha_i \underline{k(h^i, h^Y)} + b, \tag{6}$$

$$\underline{f}(Y) = \sum_{\{\alpha_i > 0\}} \alpha_i \underline{k(h^i, h^Y)} + \sum_{\{\alpha_i < 0\}} \alpha_i \overline{k(h^i, h^Y)} + b, \tag{7}$$

where h^Y denotes the interval-valued histogram of all histograms that occur for elements of Y, and $\overline{k(h^j, h')}$ and $\underline{k(h^j, h')}$ are upper and lower bounds of the kernel function itself. Table 1 lists examples of these. Note that the functional form given there is not necessarily the fastest way to compute the values. See, *e.g.*, [10] and [13] on how to speed up the evaluation of linear kernels and the histogram intersection kernel, respectively.

5. Experimental Evaluation

We evaluate the performance of ESC compared to single stage detection and to ordinary, *i.e.* exhaustive, cascade

[1] Only the function values that actually occur must be non-negative. Equations (6) and (7) hold, *e.g.*, also for SVMs with linear kernel, because the histograms that we assume as input have no negative entries.

	kernel function $k(h, h^y)$	upper bound $\overline{k(h, h^Y)}$	lower bound $\underline{k(h, h^Y)}$
linear	$\sum_j h_j h_j^y$	$\sum_j h_j \overline{h}_j^Y$	$\sum_j h_j \underline{h}_j^Y$
histogram intersection	$\sum_j \min(h_j, h_j^y)$	$\sum_j \min(h_j, \overline{h}_j^Y)$	$\sum_j \min(h_j, \underline{h}_j^Y)$
χ^2	$-\sum_j \chi^2(h_j, h_j^y)$ with $\chi^2(h_j, h_j^y) = \frac{(h_j - h_j^y)^2}{h_j + h_j^y}$	$-\sum_j \underline{\chi^2(h_j, h_j^Y)}$ with $\underline{\chi^2(h_j, h'_j)} = \begin{cases} \frac{(h_j - \underline{h}_j^Y)^2}{h_j + \underline{h}_j^Y} & \text{for } h_j < \underline{h}_j^Y, \\ \frac{(h_j - \overline{h}_j^Y)^2}{h_j + \overline{h}_j^Y} & \text{for } h_j > \overline{h}_j^Y, \\ 0 & \text{otherwise.} \end{cases}$	$-\sum_j \overline{\chi^2(h_j, h_j^Y)}$ with $\overline{\chi^2(h_j, h'_j)} = \max\left(\frac{(h_j - \underline{h}_j^Y)^2}{h_j + \underline{h}_j^Y}, \frac{(h_j - \overline{h}_j^Y)^2}{h_j + \overline{h}_j^Y}\right)$
χ^2-RBF	$\exp\left(-\gamma \sum_j \chi^2(h_j, h_j^y)\right)$	$\exp\left(-\gamma \sum_j \underline{\chi^2(h_j, h_j^Y)}\right)$	$\exp\left(-\gamma \sum_j \overline{\chi^2(h_j, h_j^Y)}\right)$

Table 1. Bounding functions for commonly used kernel functions. h_j^y denotes the j-th bin of the histogram representation of a region y. $\overline{h}_j^Y$ and $\underline{h}_j^Y$ denotes the large and smallest possible values that h_j^y can take for any $y \in Y$. For unnormalized bag-of-word histogram, these are the histograms of the largest and small rectangle in Y; for normalized histograms, they can be computed using the method of [9].

evaluation. We have performed extensive experiments on the PASCAL VOC 2006 [3] dataset that consists of 5304 image of natural scenes. The dataset contains manual annotation of the locations of all objects from 10 object classes in form of bounding boxes. In all experiments we used the *train* and *val* parts of the data for model selection. Afterwards, we retrained the models on all of *trainval* and we report results on the *test* part. To avoid a selection bias we include all object categories in our study, including those for which the chosen bag of visual word representations is known to be inferior to methods based on prototypes [1], edge-orientation [2, 5], or shape [6], for example.

5.1. Image Representation

We preprocessed each image into a set of keypoint localization with assigned integer cluster index. We identified 5,000–10,000 keypoints per image at Harris-Laplace locations as well as on a regular grid. We extracted WSIFT descriptor [18] at these locations and quantized them using a 512 entry codebook that was obtained by k-means clustering from a random subset of the descriptors. All classifiers in the following sections work with *bag of visual words* representation, *i.e*. regions are represented by the histograms of cluster indices of the keypoints that fall into them.

Note that a detection system based on this representation is unlikely to beat the state-of-the-art in object detection. Multiple publications over the last years have shown that detection performance is improved by the use of densely samples feature points and large codebooks [14], multiple sets of feature descriptors [19], spatial pyramid representations [12], and the use of image context [17].

We chose the simple setup because in this paper we do not argue in favor of a new object detection method, but we rather show the benefits of a divide-and-conquer strategy for cascade evaluation, in contrast to an exhaustive evaluation or to single-stage setups. We therefore believe that reproducibility is more important than highest overall accuracy. The use of a very compact representation and classifiers with few meta-parameters enables us to release the pre-processed feature data together with our source code[2].

5.2. Cascade Setup

For each of the 10 object categories we trained a detection cascade of 1, 2, 3, 5 and 10 linear SVMs using the procedure described in Section 3. For each stage, three bootstrapping iterations were performed to collect additional false detections. For the experiments on classification speed, we additionally trained an SVM with χ^2-RBF kernel as a nonlinear stage to put after the linear ones. We fixed the kernel's bandwidth parameter to the inverse of the mean of the χ^2-distances within the training set and we used 10-fold cross validation to determine the C parameter. To make the training and evaluation of a large number of cascades tractable, we quantize the image coordinates to multiples of 8 for the linear, and 16 for the nonlinear experiments. During cascade evaluation, we activate a score insensitivity of $\varepsilon = 0.25$ and we adapted the threshold of the exhaustive evaluation to ε to ensure a fair comparison. When outputting multiple detections per image, we used greedy non-maximum suppression with an overlap threshold of 0.8.

5.3. Evaluation of Detection Quality

Several methods to evaluate the quality of a multi-stage object detection system have been proposed. We follow Vedaldi *et al*.'s [19] setup of measuring *recall vs. overlap*. Object detection is seen as a regression-like problem in this case, where the quality of a method is judged by how well it predicts the coordinates of the ground truth object bounding boxes in previously unseen test images.

Recall–overlap evaluation has the advantage that it requires only a *local ranking* of the detected regions within an image, not a *global ranking* of detections between images. The latter kind is required by retrieval-based measures, in

[2]available at `http://www.christoph-lampert.org`.

particular *average precision (AP)* that is in the PASCAL VOC challenges. Although not directly related to the object positions, the global ranking is known to have a strong influence on the AP score when most image in the datasets do not show the target class.

The details of our evaluation procedure are as follows: for each test image containing the object class, we let the detection systems extract a fixed number of candidate images. For each ground truth region y, we then calculate its maximal overlap with any of the returned regions y', using the *area overlap* measure, $\Delta(y, y') = \frac{area(y \cap y')}{area(y \cup y')}$. From the results we derive overlap–recall curves: for any threshold, $\theta \in [0, 1]$, we compute the fraction of ground truth boxes which are identified with at least overlap θ, and plot it against the threshold value. Figure 2 shows the resulting plots for the two classes *bicycle* and *cat*. Table 2 summarizes the results in more compact form, reporting the *area under curve* for all classes and cascade setups.

Overall, the results show that cascades of linear classifiers are indeed able to improve detection quality over single stage classifiers. For all classes, longer cascades result in stronger overlap with the ground truth locations, provided that enough candidate regions from each image are taken into account. However, if only few regions from each image are used, the picture from our experiments is more diverse: for the classes *car*, *cat*, and *dog*, accuracy still grew monotonically with the cascade length. For the other classes, a single stage classifier provided as good or better overlap with the ground truth as a cascade. We believe that this is not an indicator that the cascade training has failed, since at high recall levels, the ground truth is recovered. We rather believe that the local ranking of regions is better in the first stage, which is trained directly with ground truth detections versus background regions, than in the later stages that are trained to suppress false positives. The results for the *cow* and *person* classes, show a limitation of the rather simple training procedure we used. If the classifier learned in one stage differs only insignificantly from the previous one, the training set does not change and all later stages remain practically identical. Enforcing diversity between the stages could be a way to overcome this problem.

5.4. Evaluation of Detection Speed

The previous section established what we already assumed true: that cascades even of simple linear classifiers are able to improve the detection quality over single-stage sliding window systems. In the following, we will show that using ESC we are able to evaluate such cascades more efficiently than before. In all cases we measure the performance of different three setups of cascade evaluation: ordinary per-region evaluation *(baseline)*, plain ESC as described in Section 2.1 *(ESC w/o ESS)*, and ESC with integrated ESS stage from Section 2.3 *(ESC)*. Each setup was applied to all cascades of the previous section with and without an additional nonlinear stage. To avoid variations due to different hardware platforms, all timing experiments were run on the same 2.8 GHz Linux PC. For keeping the computation manageable, we selected 100 random test images and evaluated the classifiers of all 10 classes only on these, thereby performing 1000 timing runs per cascade.

# stages	standard cascade	ESC w/o ESS	ESC
1	0.59 ± 0.01	0.58 ± 0.01	0.04 ± 0.01
2	0.51 ± 0.01	0.45 ± 0.01	0.44 ± 0.01
3	0.52 ± 0.01	0.45 ± 0.01	0.45 ± 0.01
5	0.53 ± 0.01	0.46 ± 0.01	0.46 ± 0.01
10	0.58 ± 0.01	0.49 ± 0.01	0.49 ± 0.01

Table 3. Mean runtime [in s] and standard deviation of the mean over 1000 evaluations of linear cascades.

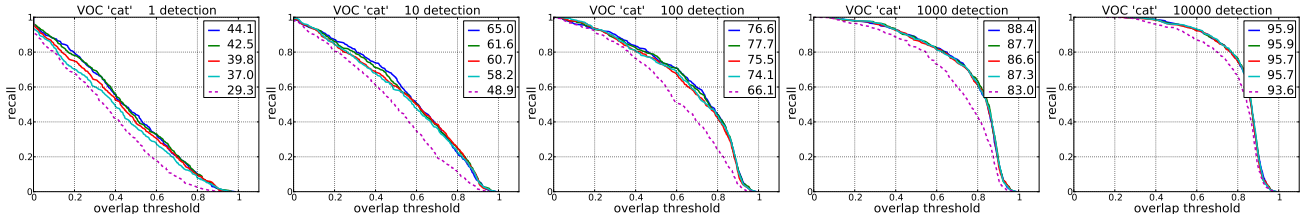
# stages	standard cascade	ESC w/o ESS	ESC
0+1†	1889 ± 104	949 ± 115	204 ± 19.6
1+1†	469 ± 34.5	346 ± 31.9	221 ± 10.0
2+1	134 ± 6.81	117.5 ± 7.37	66.1 ± 3.28
3+1	95.0 ± 6.35	83.2 ± 6.37	45.3 ± 2.31
5+1	85.4 ± 6.09	73.6 ± 6.07	40.3 ± 2.20
10+1	80.2 ± 5.90	68.3 ± 5.80	38.7 ± 2.16

Table 4. Mean runtime [in s] and standard deviation of the mean over 1000 evaluations of cascades with nonlinear last stage. (†: runtime estimated from 250 evaluations).

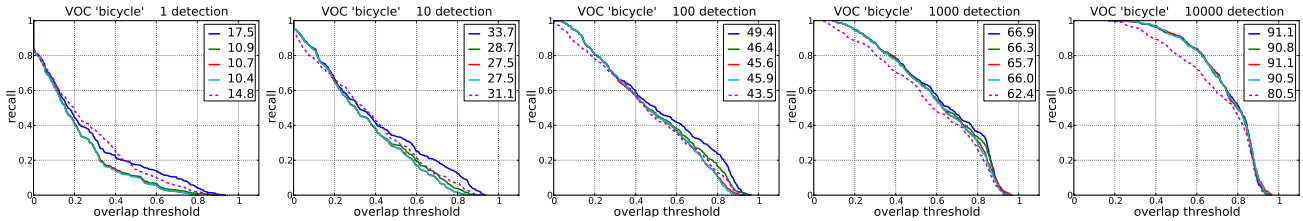
Table 3 and 4 shows the mean runtime and the standard deviation of the mean. Table 5 and 6 show the number of classifier or bound evaluations that were performed in each stage during these experiments. While the absolute runtime is more relevant for practical applications, the evaluation count better reflects the differences between the methods, because it is independent of implementational choices, and excludes the time required for preprocessing, memory management and file I/O.

The main observation from Table 3 is that for linear cascades, the runtime does not depend much on the length of the cascade. Consequently, we can turn an existing linear detector into a cascade without computational cost at evaluation time. Table 5 shows the reasons for this: most classifier evaluations occur in the first one or two cascade stages. Additional stages reduce the number of candidate regions further, but they contribute little to the overall computing time. When all subclassifiers are fast, a large part of the overall runtime consists of overhead due to preprocessing and memory management. Nevertheless, the results show a small but significant speed advantage of ESC over the exhaustive evaluation in all setups.

For cascades with a nonlinear last stage the results differ dramatically from the purely linear case. Table 4 shows that all methods are much faster when applied to longer cascades in comparison to shorter ones. Table 6 explains why: the more linear stages we use, the more candidate regions are rejected instead of ending up at the final, nonlinear stage. Since the nonlinear classifier evaluations are much slower



(a) Well-trained cascade (*cat*). All cascaded classifiers improve the overlap with the ground truth for all detection levels.



(b) Problematic cascade (*bicycle*). When extracting a large number of detections from each image (100 to 10000), all cascade setups improve the detection quality. At small detection numbers (1 or 10 detections per image) the 2 to 5-stage cascades *decrease* the detection accuracy compared to a single stage detector, which is only overcome when using the longest, 10-stage cascade.

Figure 2. Overlap *vs*. recall plots for PASCAL VOC 2006 *cat* and *bicycle* categories (best viewed in color). In each plot, the x-axis denotes the overlap threshold, the y-axis the fraction of identified ground truth boxes. Each row contains plots for 1, 10, 100, 1000 and 10000 detections per image (left to right). Each plot shows from top to bottom the curves for cascades of length 10, 5, 3, 2 and 1 (single stage classifier; dashed). Numeric values are the recall percentage at 50% overlap, which is the threshold used in the VOC evaluation procedure.

	1	10	100	1000	10000	1	10	100	1000	10000	1	10	100	1000	10000	1	10	100	1000	10000	1	10	100	1000	10000
1	24.9	35.9	46.1	56.9	70.2	19.4	28.0	39.8	51.8	67.9	13.6	21.3	30.4	42.1	58.7	34.8	47.1	58.9	69.8	78.9	15.8	24.6	35.0	48.2	64.5
2	19.9	33.7	47.0	60.6	74.9	15.1	27.3	41.6	58.4	72.5	14.4	24.1	34.6	46.5	63.9	40.1	54.5	66.8	76.4	82.4	12.3	23.1	36.7	52.5	70.6
3	20.0	33.7	46.9	61.0	75.0	14.8	27.3	41.9	58.5	72.6	17.5	26.6	36.0	47.1	63.1	42.6	55.3	66.3	76.2	82.0	12.3	23.3	36.8	52.5	70.8
5	20.5	34.7	48.2	61.5	74.9	14.8	27.3	41.9	58.4	72.6	23.1	29.4	37.6	47.8	62.7	44.7	56.1	67.6	76.7	82.3	12.3	23.3	36.9	52.5	70.8
10	25.0	38.5	50.7	62.4	75.0	14.8	27.3	41.8	58.8	73.1	25.2	31.6	39.4	49.6	62.8	44.7	56.7	67.9	76.4	82.3	12.3	23.3	36.9	52.5	70.8
	bicycle					**bus**					**car**					**cat**					**cow**				
1	29.4	40.7	52.7	65.4	76.6	21.5	31.7	43.3	55.0	69.7	24.7	36.1	47.6	59.3	71.2	7.9	14.4	22.9	36.3	55.6	12.0	18.4	26.7	37.9	54.7
2	27.9	42.6	56.6	70.9	80.3	15.1	28.7	45.7	61.8	75.3	18.2	33.8	48.4	61.4	75.8	5.8	13.0	24.7	41.7	58.7	8.9	17.3	28.2	43.9	62.8
3	30.7	44.3	58.1	71.1	80.0	15.2	28.6	45.7	61.8	75.2	18.2	33.8	48.4	61.3	75.6	5.8	12.9	24.6	41.5	56.2	8.7	17.4	28.3	44.0	62.6
5	33.8	47.1	60.1	71.5	80.0	15.2	28.7	46.1	62.2	75.2	18.4	33.8	48.5	62.1	76.1	5.8	12.9	24.6	41.5	56.1	8.7	17.4	28.3	43.9	63.0
10	37.7	49.0	61.2	71.8	79.7	16.4	30.3	47.5	63.7	75.4	19.4	34.8	49.8	63.4	76.0	5.8	12.9	24.6	41.5	56.1	8.7	17.4	28.4	44.2	63.6
	dog					**horse**					**motorbike**					**person**					**sheep**				

Table 2. Area under recall–overlap curve [in %] for the top 1, 10, 100, 1000 and 10000 detections of cascades of length 1, 2, 3, 5 and 10.

than linear ones, it is their number that dominates the overall runtime. The tables also show that ESC – in particular in combination with ESS – is more effective in avoiding nonlinear evaluations, thereby providing a significant speedup over exhaustive evaluation. When applied to identical classifier cascades, ESC is always at least twice as fast.

6. Summary and Outlook

In this paper we have introduced ESC, a divide-and-conquer strategy for accelerating the evaluation of classifier cascades for object detection in natural images. By using an internal representation by set of regions instead of individual regions, ESC can discard large fractions of the potential candidate locations with few classifier evaluations. Thereby it reduces the computational effort compared to the standard way of cascade evaluation for object detection, in which one applies the classifier cascade exhaustively to every candidate region in the images. In our experiments, this resulted in a speedup of approximately 15% in the case where all stages are linear and over 50% in the nonlinear case.

One reason for this effect is that ESC allows the integration of the branch-and-bound based ESS algorithm [10] in its final stage, because both rely on the same internal representation. ESC in this way combines the advantages of two current trends for fast object detection: global optimization techniques that exploit spatial correlation of the detection scores and cascades that provide a speedup by approximating the actual detection function with increasing precision.

An important problem that we were able to address only superficially in this paper is the question of how to best learn a classifier cascade that allows efficient evaluation with ESC. Because ESC benefits from smoothly varying decision functions, we conjecture that it would be beneficial to enforce strong regularization in the first stages, whereas later stages that have less influence on the overall runtime can be more specifically tuned to the data. This resembles the concept of hierarchical multiclass classification, and we plan to explore this relation by extending ESC to

method	# stages	1	2	3	4	5	6	7	8	9	10	**total**
ESC [w/o ESS]	1	0.7 [32.0]	–	–	–	–	–	–	–	–	–	**0.7 [32.0]**
baseline cascade	1	33.7	–	–	–	–	–	–	–	–	–	**33.7**
ESC [w/o ESS]	2	32.0	3.6 [5.3]	–	–	–	–	–	–	–	–	**35.6 [37.3]**
baseline cascade	2	33.7	10.5	–	–	–	–	–	–	–	–	**44.2**
ESC [w/o ESS]	3	32.0	5.3	1.02 [1.64]	–	–	–	–	–	–	–	**38.3 [38.9]**
baseline cascade	3	33.7	10.5	2.49	–	–	–	–	–	–	–	**46.7**
ESC [w/o ESS]	5	32.0	5.3	1.64	1.19	0.62 [1.10]	–	–	–	–	–	**40.7 [41.2]**
baseline cascade	5	33.7	10.5	2.49	1.67	1.55	–	–	–	–	–	**50.0**
ESC [w/o ESS]	10	32.0	5.3	1.64	1.19	1.10	1.07	1.05	1.04	1.02	0.58 [1.02]	**46.0 [46.4]**
baseline cascade	10	33.7	10.5	2.49	1.67	1.55	1.49	1.47	1.45	1.43	1.42	**57.2**

Table 5. Classifier or bound evaluations [in 10^5] for linear classifiers cascades. Results in brackets are without integration of ESS into ESC.

method	# stages	1	2	3	4	5	6	7	8	9	10	**nonlinear**
ESC [w/o ESS]	0+1	–	–	–	–	–	–	–	–	–	–	**28.65 [137.4]**
baseline cascade	0+1	–	–	–	–	–	–	–	–	–	–	**208.0**
ESC [w/o ESS]	1+1	244.6	–	–	–	–	–	–	–	–	–	**27.00 [53.54]**
baseline cascade	1+1	208.0	–	–	–	–	–	–	–	–	–	**64.8**
ESC [w/o ESS]	2+1	244.6	40.25	–	–	–	–	–	–	–	–	**8.03 [15.95]**
baseline cascade	2+1	208.0	64.84	–	–	–	–	–	–	–	–	**16.37**
ESC [w/o ESS]	3+1	244.6	40.25	12.84	–	–	–	–	–	–	–	**5.47 [11.18]**
baseline cascade	3+1	208.0	64.84	16.37	–	–	–	–	–	–	–	**11.20**
ESC [w/o ESS]	5+1	244.6	40.25	12.84	9.28	8.61	–	–	–	–	–	**4.90 [9.96]**
baseline cascade	5+1	208.0	64.84	16.37	11.20	10.36	–	–	–	–	–	**10.02**
ESC [w/o ESS]	10+1	244.6	40.25	12.84	9.28	8.61	8.30	8.18	8.06	7.92	7.87	**4.73 [9.35]**
baseline cascade	10+1	208.0	64.84	16.37	11.20	10.36	10.02	9.84	9.71	9.62	9.55	**9.50**

Table 6. Classifier or bound evaluations [in 10^3] for classifiers cascades with a nonlinear last stage.

other topologies than linear chains, in particular to decision trees and multi-class decision DAGs.

References

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

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

[3] M. Everingham, A. Zisserman, C. K. I. Williams, and L. Van Gool. The PASCAL Visual Object Classes Challenge 2006 (VOC2006) Results. http://www.pascal-network.org/challenges/VOC/voc2006/results.pdf.

[4] P. Felzenszwalb, R. Girshick, and D. McAllester. Cascade object detection with deformable part models. In *CVPR*, 2010.

[5] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *PAMI*, 2009.

[6] V. Ferrari, L. Fevrier, F. Jurie, and C. Schmid. Groups of adjacent contour segments for object detection. *PAMI*, 30(1), 2008.

[7] K. Fukunaga and L. D. Hostetler. The estimation of the gradient of a density function with applications in pattern recognition. *IEEE Trans. Information Theory*, 21(1), 1975.

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

[9] C. H. Lampert. Detecting objects in large image collections and videos by efficient subimage retrieval. In *ICCV*, 2009.

[10] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *CVPR*, 2008.

[11] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Efficient subwindow search: A branch and bound framework for object localization. *PAMI*, 2009.

[12] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006.

[13] S. Maji, A. C. Berg, and J. Malik. Classification using intersection kernel support vector machines is efficient. In *CVPR*, 2008.

[14] E. Nowak, F. Jurie, and B. Triggs. Sampling strategies for bag-of-features image classification. In *ECCV*, 2006.

[15] S. Romdhani, P. Torr, B. Schölkopf, and A. Blake. Computationally efficient face detection. In *ICCV*, 2001.

[16] H. Schneiderman. Feature-centric evaluation for efficient cascaded object detection. In *CVPR*, 2004.

[17] A. Torralba. Contextual priming for object detection. *IJCV*, 53(2), 2003.

[18] K. E. A. Van De Sande, T. Gevers, and C. G. M. Snoek. Evaluation of color descriptors for object and scene recognition. In *CVPR*, 2008.

[19] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman. Multiple kernels for object detection. In *ICCV*, 2009.

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

[21] T. Yeh and T. Darrell. Fast concurrent object localization and recognition. In *CVPR*, 2009.

[22] J. Yuan, Z. Liu, and Y. Wu. Discriminative subvolume search for efficient action detection. In *CVPR*, 2009.

[23] Q. Zhu, S. Avidan, M. C. Yeh, and K. T. Cheng. Fast human detection using a cascade of histograms of oriented gradients. In *CVPR*, 2006.