An Efficient Shared Multi-Class Detection Cascade

Philipp Zehnder, Esther Koller-Meier, and Luc Van Gool
Computer Vision Laboratory, ETH Zurich, Switzerland
{zehnder,ebmeier,vangool}@vision.ee.ethz.ch

Abstract

We propose a novel multi-class object detector, that optimizes the detection costs while retaining a desired detection rate. The detector uses a cascade that unites the handling of similar object classes while separating off classes at appropriate levels of the cascade. No prior knowledge about the relationship between classes is needed as the classifier structure is automatically determined during the training phase. The detection nodes in the cascade use Haar wavelet features and Gentle AdaBoost, however the approach is not dependent on the specific features used and can easily be extended to other cases. Experiments are presented for several numbers of object classes and the approach is compared to other classifying schemes. The results demonstrate a large efficiency gain that is particularly prominent for a greater number of classes. Also the complexity of the training scales well with the number of classes.

1 Introduction

Substantial progress has been made recently in object detection. But most of this work focuses on detecting instances of one particular object against the background. Obviously, such solutions can be extended to multi-class detection, i.e. determining where in the images is any of the target classes of objects, by repeating a search for each of those classes separately. The disadvantage of such strategy is that time goes up quickly with the number of target classes. An advantage is that such process results in the simultaneous distinction between target classes. At the other end of the spectrum, one can build a system that tries to distinguish any of the object classes from background within one and the same process. This may be difficult, as different target classes and background may be interspersed in feature space. Moreover, in case one wants to know about the specific class of each object, a second step would be required. Here, we propose an intermediate solution, which runs in sub-linear time with regard to the number of classes by heavily exploiting the sharing of features for the detection of the different classes. It also to a large extent provides a distinction between the classes, requiring minor further classification effort for a full such analysis compared to the previous case. The three types of cascades are illustrated in Fig. 1. In summary, whereas the first type of approach quickly leads to unacceptable computation times and the second tends to be performing poorly, the third kind of mixed strategy that we propose avoids such weaknesses.

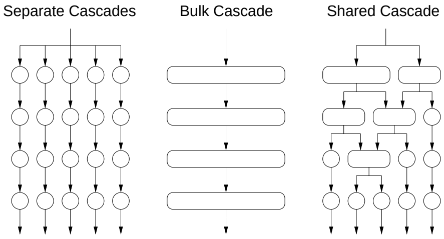

Figure 1: *Types of cascades.*

Work in this direction has been presented by Torralba et al. [8], focusing on sharing features between object classes in a boosting framework. They propose a code matrix approach associating features and classes, with an emphasis on joint learning of good features and subsets of classes. The work of Opelt et al. [7] proposes learning an alphabet of boundary fragments that can be shared among classes. Apart from the different features, it allows incremental learning and control over the degree of sharing. Blanchard and Geman [1] explore the recognition of many pattern classes by sequences of *tests* in a hierarchical framework. They prove the optimality of coarse-to-fine searching under very general conditions. Apart from class relationships, this also applies to cascaded background rejection, but the issue of learning the classifier structure is left open.

Our proposed approach combines multi-class handling and feature sharing with the efficiency of cascade approaches. Instead of separate cascades for each class or a bulk cascade for all classes we build a shared cascade as in Fig. 1. In contrast to other multi-class cascade approaches, where the structure is designed by hand (e.g. Huang et al. [4]), our approach builds the structure completely automatically. Moreover, note that our shared cascades amount to structures beyond mere hierarchies (see Fig. 1). Another advantage of our approach is that a specific class only uses a subset of features (in contradistinction to e.g. [6]) and that some features are shared between classes to further reduce computation time. We discuss two versions of this. In the first we iteratively build up the shared cascade, growing the cascade stage by stage. In the second we follow a greedy approach that fixes the overall cascade structure in one go.

2 Algorithm

We start with explaining the iterative version of our shared cascade building algorithm. It is illustrated in Fig. 2. Each iteration consists of three basic steps, (i) training of single-class detector nodes, (ii) determining class similarities and building groups of classes, and (iii) training a detector node for each group, replacing the previous individual nodes. Then the next layer of single-class detector nodes is added (see Fig. 2), which are then again grouped, whereafter the layer of grouped nodes is trained. Etc. Like this the shared cascade structure (Fig. 1) is built layer by layer. Henceforth we will refer to these layers as 'stages', in accordance with cascade terminology. Typically we stop when a predefined rejection rate has been reached.

(i) Training single-class detector nodes. The first step consists of training a detector node for every class individually. Such a node is designed as to let pass a minimum percentage of instances of that class, while achieving a minimum rejection rate for back-

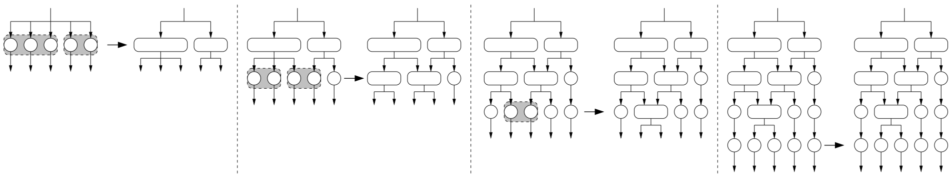

Figure 2: *Building a shared cascade. Classes with high similarity are grouped and replaced by a shared node.*

ground patterns. In our particular case, we use a combination of Haar wavelet features and AdaBoost learnt classifiers for that task. This is the same approach as in [10] with the following variations. To reduce the training complexity, we use a smaller base feature set. Instead of the exhaustive set of extended Haar-like features and full oversampling, we only use the wavelet components of size 4, 8 and 16 pixels, oversampled at 2 pixel intervals. This results in 1425 features in the 32x32 pixel reference window. For boosting, we use Gentle AdaBoost [3] because it is easy to implement, numerically robust and provides better performance than other boosting approaches [5]. Instead of weak classifiers built on simple thresholding of the feature value, we use a probabilistic classification model (see Fig. 3). It is based on the feature value histogram that delivers class likelihoods, used in GentleBoost to update the classification function. The probabilistic model is more powerful as it also allows for multi-modal distributions. At the same time, it is just as fast because determining the class likelihood consists of simply discretizing the feature value and making an access in a look-up table.

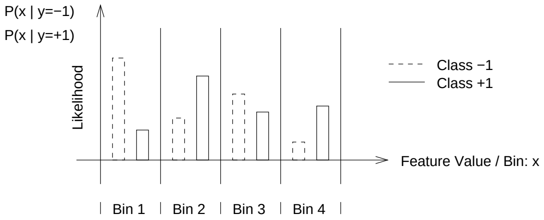

Figure 3: *Model of the weak classifier for GentleBoost. The class likelihoods given by the feature value histogram are used to update the classification function.*

(ii) Determining class similarities and building groups of classes. In step (ii), we first determine class similarities. For each class, we measure the percentage of instances that pass through at each of the other single-class detector nodes of step (i). Those percentages build a pairwise similarity matrix with entries ρ_{ij} (see Fig. 4). This is used to group classes through agglomerative hierarchical clustering (complete linkage) [2] using the distance measure

$$d_{max}(S_m, S_n) = \max_{i \in S_m, j \in S_n} (1 - \min(\rho_{ij}, \rho_{ji})) \qquad . \tag{1}$$

Groups are built only for classes with a minimum similarity, chosen here as $\rho_{ij} > 0.75$.

(iii) Training a detector node for class groups. Step (iii) consists of training a detector node for each group of classes (could still be a single-class node). The training uses the same methods as in step (i), with the same minimum detection and rejection rates.

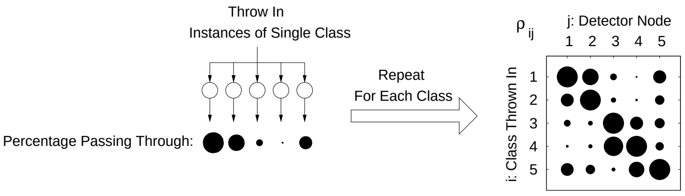

Figure 4: *Construction of the class similarity matrix.*

The algorithm described so far is the *iterative* version. As a greedy alternative, we have also tested a *flash* version, which we have found to perform equally well. There we fix the entire structure right away (see Fig. 5). This is done by first creating a full cascade for each class (not a single node layer only). Then, all object classes are injected at the top of all cascades, and it is analysed what percentage tickles down at each stage (see Fig. 6). So, the similarity matrices for the different stages are calculated all directly from this procedure and grouping is done for all stages at once. Only then, the shared cascade is trained.

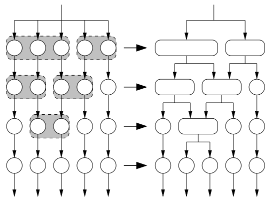

Figure 5: *Flash version of building a shared cascade.*

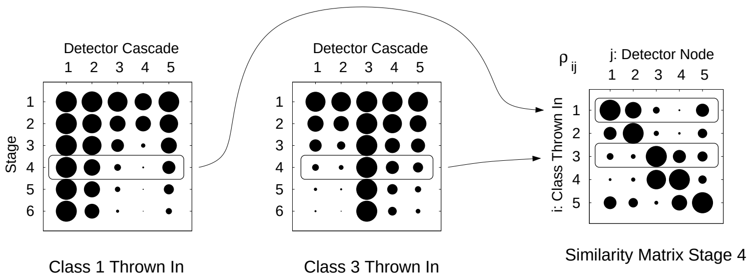

Figure 6: *Construction of the class similarity matrices in the flash version.*

The computational complexity of our approach is basically quadratic in the number of classes. However, the most time-consuming part in the conducted experiments proved to be the training of the nodes, which depends only linearly on the number of classes.

3 Experiments

The proposed approach is tested on a set of 23 real-world object classes from the MIT-CSAIL Database of Objects and Scenes [9] (the 16 classes shown in Fig. 10, plus *screenFrontal, head, chairWhole, mug, cpu, bottle, can*). We compare our *shared cascade* approach to the two straightforward approaches of using *separate cascades* for each class and using a *bulk cascade* for all classes. In particular, we investigate the relation between cost and performance, and the scalability with respect to the number of classes.

The following parameters are fixed and the same across all experiments and approaches. Each node is trained using a set of training examples of the corresponding object class(es) – 90% of all examples, the rest serving as test set – plus a bootstrap-sample of 4000 background patterns passing the preceding cascade stages. The minimum detection and rejection rates per node are set to 99% and 40% respectively. The choice of these values is somewhat arbitrary. For the rejection rate, we go out from the assumption, that often about one third of an image is something "simple" like sky or road. And that should be eliminated in the first step. So, taking into account a slightly non-optimal classifier generalization, we arrive at a value of 40%. Where not mentioned otherwise, we use the flash variant to construct the shared cascade. This is our preferred option, as it performs equally well as the iterated one, while being faster to compute. This will be demonstrated next.

The differences between the flash and the iterative construction of the shared cascade are very small and only visible when taking a closer look. Fig. 7 plots the cost, i.e. the average number of features needed, against (1 - rejection rate). The plots show two particular training runs with 23 classes. We see that one time the iterative variant is slightly better, the other time the flash variant. However, the differences are virtually negligible in view of the differences to other approaches as we will see shortly. With this comparison settled, we now fully focus on the flash version.

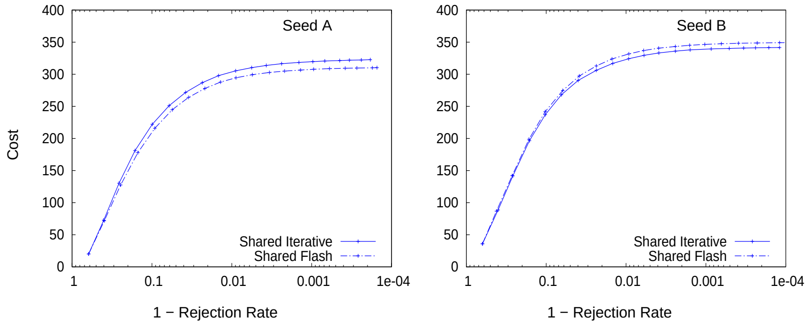

Figure 7: *Comparison of iterative and flash construction of a shared cascade.*

In a first series of experiments, we investigate the cost as a function of the performance, i.e. we determine how many features are needed on average to reach a certain background rejection rate. This evaluation is done for 4 different problem settings varying in the number of object classes (2, 4, 8, 16). For each setting, we use 5 different combinations of classes, and in the case of 23 classes, we vary the samples. This yields an average curve. For control purposes, we also determine the corresponding detection rates. The

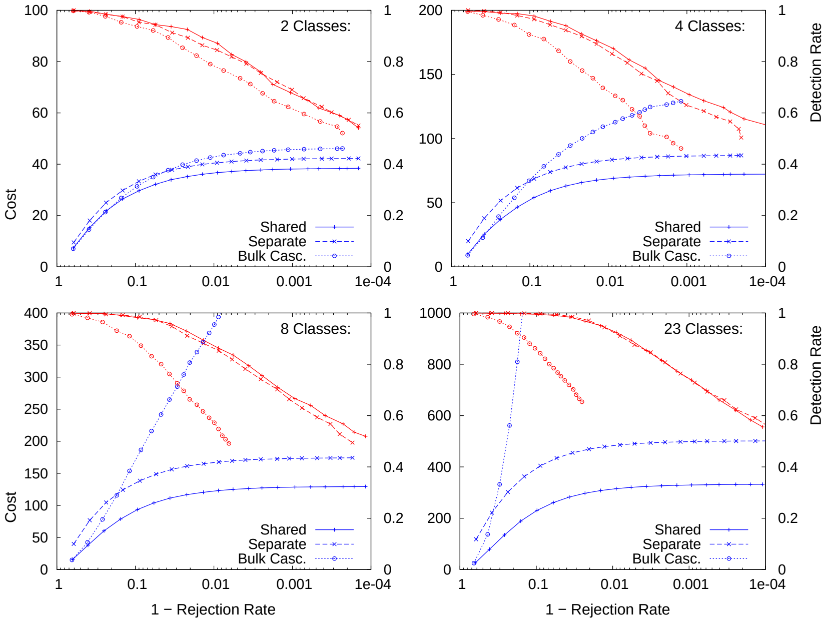

Figure 8: *Cost vs. rejection rate (blue, starting bottom-left) and detection rate vs. rejection rate (red, starting top-left) for different numbers of classes. Each marker corresponds to one stage.*

results (see Fig. 8) allow to draw a number of interesting conclusions. At low rejection rates (i.e. at the first 1-2 cascade stages), the shared approach and the bulk cascade consistently have a lower cost than separate cascades. So, merging classes at the early stages of a cascade proves to be valuable, and our algorithm does that successfully. When trying to reject more background, the cost for a bulk cascade however increases much more than for shared or separate cascades. Especially with many classes, the increase gets dramatic. The curves for shared and separate cascades rather flatten out, independent of the number of classes. The shared cascade however has a lower cost increase at any point. So, together with the lower cost in the first few stages, it is is considerably more efficient in any case. The difference is again most evident with many classes.

Another aspect to mention is the bad performance of the bulk cascade. In the case of 23 classes, its detection rate after 5 stages has already dropped to 90%, where it should theoretically be at $0.99^5 \approx 95\%$. Also, the rejection rate degrades heavily after that point. The shared approach and separate cascades do not suffer from these shortcomings up to some generalization error. After 10 stages, the detection rate quite exactly meets the target of $0.99^{10} \approx 90.4\%$, and the rejection rate of 99.1% is very close to the theoretical $1 - 0.6^{10} \approx 99.4\%$.

In a second series of experiments, we investigated the scalability with respect to the number of classes. To that purpose, we produced cost versus the number of classes graphs. As the behavior varies at different rejection rates, we did the experiments with different sized cascades, namely with 1, 3, 5 and 7 stages (corresponding to rejection rates of 40%, 78%, 92% and 97%). Fig. 9 shows the results.

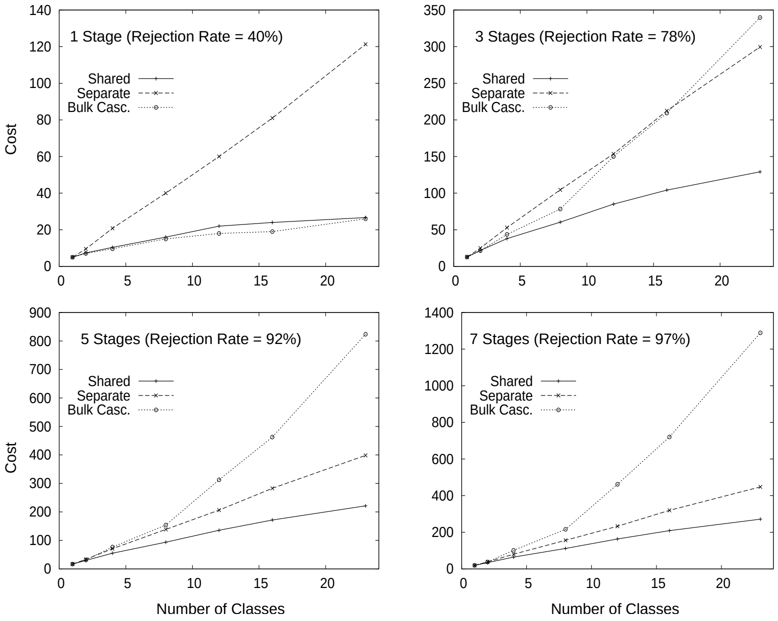

Figure 9: *Cost vs. the number of classes for cascades of different depths (numbers of stages).*

In the case of only one stage, we see that the shared and bulk cascade offer a much lower than linear behavior (which is what obviously happens with separate cascades). It resembles a logarithmic curve. Incidentally, this confirms the results found by Torralba et al. [8] for an equivalent performance level (AUROC = 0.9)[1]. When adding more stages however, the efficiency of the bulk cascade gets worse. Eventually, its cost even becomes super-linear with respect to the number of classes. This indicates that the "complete" feature sharing of a bulk cascade will turn into a major problem when growing the cascade deeper and for larger numbers of classes. In contrast, our shared approach adapts the feature sharing and exhibits a sub-linear behavior, which is considerably better than the linear cost increase with separate cascades. Moreover, individual classes are distinguished.

An interesting aspect is also the actual sharing graph produced by our algorithm. It is shown in Fig. 10 for the case of 16 classes. We see that it mainly creates a hierarchical structure. In the first stage, there are three categories that actually contain objects of similar appearance. The middle category contains long and flat objects, the right category narrow and tall objects, and the left category things that are wide and tall. Towards deeper stages, it subdivides these categories into smaller subcategories until there are only singletons left. So, from a human point of view such a structure makes sense. A peculiarity is however, that the hierarchy is not pure. There are connections of non-hierarchical nature (indicated with dashed lines). Some objects, like e.g. the stop sign, are passed through different categories. Although that might seem to cause excess cost by allowing patterns to reach some node through different paths, it is not an issue. We retain the information of what has been rejected already. Thus, if a pattern for example passes node (A) but not

[1] AUROC: Area under the ROC curve

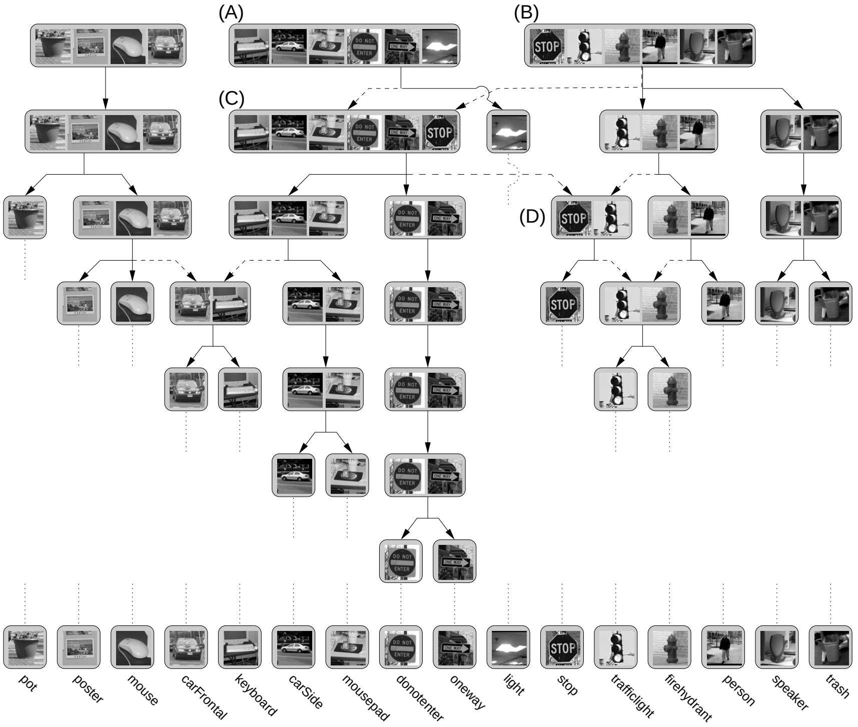

Figure 10: *Sharing graph for 16 classes. The algorithm basically creates a class hierarchy. Dashed lines indicate connections of non-hierarchical nature (see text for details about nodes (A)-(D)).*

(B), it will reach node (C). But from there it will not be passed down to (D), because all the class hypotheses of (D) have already been rejected by (B).

Finally, we provide a visualization of the detection process for the case when we limit the number of features. Fig. 11 shows how far we can get if the cost is fixed to levels of about 80, 200 and 250 features on average. We plot the number of remaining hypotheses for each pixel (brighter=more hypotheses, black=completely rejected). A separate graphic shows only the parts for a specific single object class hypothesis. This is done for the classes *carSide* and *stopSign*, as they actually appear in the image. In order to avoid obfuscation from the overlay of different scale levels, we analyse them separately.

The results show that, for a given computation power, we can achieve a much higher background rejection rate with the shared approach, compared to the other approaches. The separate cascades perform very poorly at completely rejecting background patterns, even if such patterns get rejected by a majority of single-class cascades. This is visible from the overall darker shade or from the class specific plot. In the per-class sense, separate cascades and the bulk cascade finally reach about the same performance level in throwing out background. Yet, this is by far too bad for a usable detection, in contrast to the shared approach that leaves only a small number of false positives. With an additional postprocessing step that merges overlapping detections and applies a threshold we get good detection results.

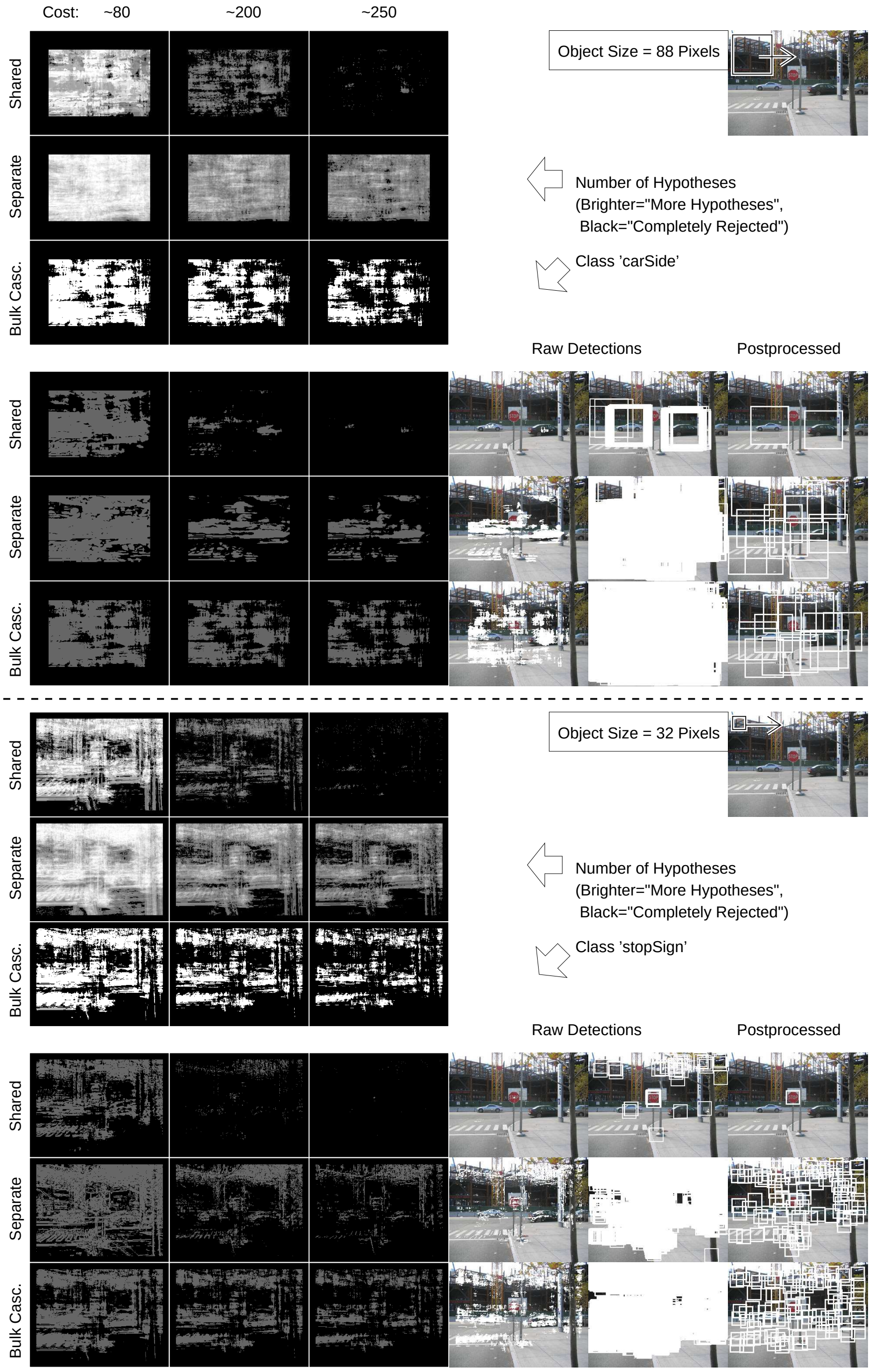

Figure 11: *Results when running a 16 class detector on a full image. Comparison of different cascade concepts (shared, separate cascades, bulk cascade) for different cost levels, i.e. when computation power is limited.*

A so far unexplored advantage of our shared cascade scheme is that is amenable to the gradual addition of extra classes. It can be expected that as soon as a certain set of classes has been added, a new class will be similar to a limited set of those. The new class is injected into the existing shared cascade, where it will only penetrate along certain paths (following the track of similar classes already present). Then only these parts of the cascade need to be adapted.

4 Conclusion

We have investigated a shared multi-class detection cascade, combining joint and separate handling of object classes. Comparing this to separate cascades or a bulk cascade, we have observed considerable efficiency gains and the advantage increases with the number of classes. The automatically created sharing structure is meaningful from a human point of view and is more general than a simple hierarchy. Running the detector on real images with a limited investment of computation power shows large improvements. When separate cascades or a bulk cascade only give unusable results, our shared approach delivers good detections. Furthermore, the training complexity is very low.

Acknowledgment – Part of this work was funded through Swiss NCCR project IM2.

References

[1] G. Blanchard and D. Geman. Hierarchical testing designs for pattern recognition. *Annuals of Statistics*, 33(3):1155–1202, 2005.

[2] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, November 2000.

[3] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. Dept. of Statistics, Stanford University Technical Report, 1998.

[4] Chang Huang, Haizhou Ai, Yuan Li, and Shihong Lao. Vector boosting for rotation invariant multi-view face detection. In *ICCV '05*, pages 446–453, 2005.

[5] R. Lienhart, A. Kuranov, and V. Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *DAGM-Symposium 2003*, pages 297–304.

[6] Jim Mutch and David G. Lowe. Multiclass object recognition with sparse, localized features. In *CVPR '06*, pages 11–18.

[7] Andreas Opelt, Axel Pinz, and Andrew Zisserman. Incremental learning of object detectors using a visual shape alphabet. In *CVPR '06*, pages 3–10.

[8] A. Torralba, K. P. Murphy, and W. T. Freeman. Sharing features: efficient boosting procedures for multiclass object detection. In *CVPR '04*, pages 762–769.

[9] A. Torralba, K. P. Murphy, and William T. Freeman. MIT-CSAIL database of objects and scenes. URL: http://web.mit.edu/torralba/www/database.html.

[10] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR '01*, pages 511–518.