

# Efficient, Simultaneous Detection of Multiple Object Classes

Philipp Zehnder, Esther Koller-Meier, and Luc Van Gool  
Computer Vision Laboratory, ETH Zurich  
Sternwartstrasse 7, 8092 Zurich, Switzerland  
{zehnder,ebmeier,vangool}@vision.ee.ethz.ch

## Abstract

*At present, the object categorisation literature is still dominated by the use of individual class detectors. Detecting multiple classes then implies the subsequent application of multiple such detectors, but such an approach is not scalable towards high numbers of classes. This paper presents an alternative strategy, where multiple classes are detected in a combined way. This includes a decision tree approach, where ternary rather than binary nodes are used, and where nodes share features. This yields an efficient scheme, which scales much better. The paper proposes a strategy where the object samples are first distinguished from the background. Then, in a second stage, the actual object class membership of each sample is determined. The focus of the paper lies entirely on the first stage, i.e. the distinction from background. The tree approach for this step is compared against two alternative strategies, one of them being the popular cascade approach. While classification accuracy tends to be better or comparable, the speed of the proposed method is systematically better. This advantage gets more outspoken as the number of object classes increases.*

## 1. Motivation

Detecting objects from many different classes in images is still a challenging task. This is especially true if these objects can appear against cluttered backgrounds. One is confronted with the following main difficulties: a) the huge number of locations and scales at which an object may appear, b) the usually wide but unknown variety of patterns in the background, and c) the overall variability of the object classes to be detected.

At present, the object categorisation literature is still dominated by the use of individual class detectors, e.g. a detector only for cars, another one for pedestrians, etc. Detecting multiple classes then implies the subsequent application of multiple such detectors. This approach does not scale well towards high numbers of classes.

Faced with these challenges, we propose a single scheme, that detects several classes in an interwoven fashion. It takes the form of a decision tree. Its computation time goes up much less than linear with the number of object classes to be detected. Before describing the construction of such trees, it is useful to list some observations which have motivated their design.

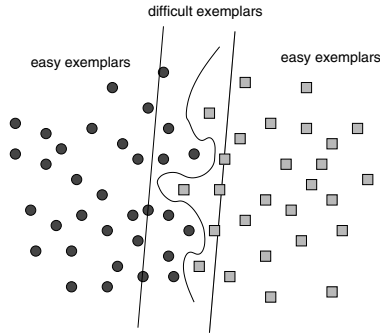
**(O1) Unbalanced Datasets:** Without prior knowledge, object detection involves a brute-force search over the whole image. A large number of windows have to be checked for the presence of a target object. In practice, this number is in the range of several thousands, whereas the number of object instances is hardly ever more than a few dozen. Hence, the overall processing cost is mostly determined by the cost of processing background patterns.

**(O2) Richness of the Background Class:** In addition to the problem of unbalance, the background patterns may consist of just “everything else which is not a target object”. Hence, the background may include objects of classes closest in appearance to the target classes. Also, the background patterns will be far from clustered in any feature space. That makes the separation of an object class from the background harder than between target object classes.

**(O3) Shared features:** More often than not, real life object classes show high variability among their instances and lack unique discriminative features. Multiple classes may share certain features, which nevertheless puts them apart from many other classes without these features.

**(O4) Overlapping distributions:** Telling classes apart is often made difficult by the overlap in their feature distributions. Basically, patterns can be split into subsets of ‘easy’ and ‘difficult’ patterns, see fig. 1. Moreover, what is easy or difficult depends on the choice of features. A more sophisticated classifier – using more intricate or higher numbers of features – can make more correct decisions than a simple one, at the expense of its higher computational cost.

These observations have led us to several design decisions. Given that classes often share features (O3) a decision tree approach stands to reason. It leads to the gradual



**Figure 1. Most patterns can be classified by a simple classifier if a third, fuzzy region is kept undecided.**

subdivision of patterns into smaller sets of classes, based on joint decisions at the nodes. Rather than using binary decisions at the nodes of the tree, as is usually done, we use ternary decisions. The inspiration comes from (O4). Patterns are split into three subsets: two that correspond to clearcut cases that should go one or the other direction down the branches of the tree (i.e. the ‘easy’ patterns), and a third route for the less outspoken cases (i.e. the ‘difficult’ patterns) for which decisions are postponed until analysed further.

Another important decision we took is to split the detection of objects from multiple target classes into two stages. The first aims at splitting off the background, so that it can be discarded in the second phase, which tells the targeted object classes apart. This special treatment of the background class is warranted by (O1) and (O2). In this paper, we focus on the first stage exclusively, given the available space. The paper proposes a strategy where the object samples are first divided into sets that facilitate their distinction from the background.

It is useful to note that the first stage already delivers a very useful tool. It will detect the presence of any of a series of object types that are of interest. For instance, in a surveillance application, it may be important to detect an intruder, whether s/he is on foot, in a car, or driving a motorbike. The important thing is to detect all such instances, not to distinguish them, and then hand over to a human operator to decide what to do.

## 2 Relation to the state-of-the-art

The approach presented here is based on the theory of decision trees like CART/ID3/C4.5 [3, 7, 8], and integrates the principle of cascade classifiers [12, 4], shared features [11] and the design of hierarchical multi-class detectors [2, 1]. But we go beyond these important, original contributions.

Joint boosting [11] reduces the total number of features needed for the detection of multiple classes in comparison to separate, independently trained detectors. It relies on the actual existence of shared features. However, all the features are still calculated for all the classes. Our approach is aimed at only calculating those needed for the handling of a pattern. This parsimonious approach leads to higher efficiency and robustness, given the curse of dimensionality.

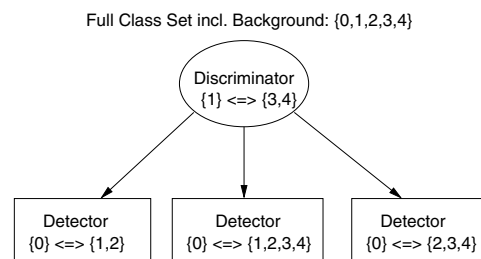
In [2] a coarse-to-fine approach is described for multi-class detection. They propose a recursive splitting of the classes into mutually exclusive subsets. Yet, such a strictly hierarchical scheme of ever finer partitions cannot be taken for granted in general. More complex dependencies among classes are to be expected. In our approach, we allow for splits that let patterns of the same class follow different branches.

We use Haar wavelets as features, extracted from fixed-sized windows which are shifted over the entire image. Feature selection accelerates the detection and also gives a rough segmentation of the objects. This has been inspired by successful earlier work [5, 12, 4]. For the classifiers at the nodes we use Support Vector Machines [9].

## 3. Learning algorithm

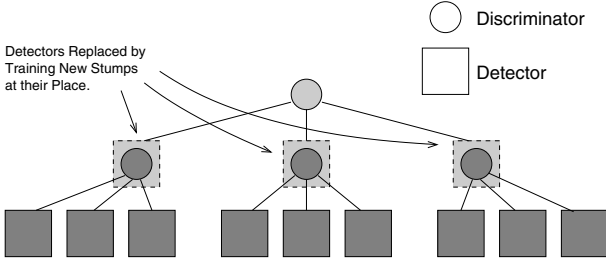
### 3.1. Tree stumps as recursive atoms

We build our trees in a recursive fashion. The building atom is the so-called ‘stump’, shown in fig. 2. It consists of two layers. The first is the *discriminator*, which tries to split the input classes into subsets of these. The second layer consists of *detectors*, which try to tell apart their share of the object patterns from their background patterns. As can be seen in the figure, the discriminator makes a ternary decision, with patterns being directed towards a left, middle, and right branch. The left and right branch are where the ‘easy’ patterns of (O4) in section 1 are collected, and the middle branch is the repository for the ‘difficult’ ones.



**Figure 2. Stump: Simple decision tree atom consisting of a discriminator and 3 detector nodes.**

The construction of the tree proceeds iteratively. After deciding on the classes split by the discriminator, and the features on which this split would have to be based, each of the detectors are replaced by a new stump, to be designed next. This process of subsequent detector expansion is illustrated in fig. 3.



**Figure 3. Tree growth by stump iteration.**

This way, the tree has been grown to the full as soon as no further expansions are carried out. The detectors remaining at that point represent its leaves, and the discriminators generated throughout the iteration procedure yield the other nodes. In this paper, we develop three layers of stumps maximum (i.e. the maximum tree ‘depth’ is 3), after which the iteration stops. Hence, the detectors of the 3rd generation of stumps yield the leaves of the tree. Of the 1st and 2nd generation stumps only the discriminators remain.

### 3.2. Tree growth

In order to build the tree, a greedy best-first search method is used as in traditional tree learning algorithms. That is, the tree is built top-down by training one stump at a time. For each discriminator, to become a node in the final tree, several candidate splits are evaluated and the best one is inserted. The final nodes will recursively split up the given class sets into smaller subsets, as usual. However, our method differs from standard approaches in several ways:

- Our final goal is to optimize the detection speed by splitting up class sets into smaller subsets that can be better separated from the background. We do not optimize some impurity measure (e.g. entropy as in ID3 [7] and C4.5 [8]), but base our decision for the best split candidate on the minimization of the mean detection cost in terms of computation. In fact, we look for a split-and-detect combination such that the overall detection cost is reduced as much as possible. More detail is found in section 3.4
- Instead of testing single features individually as in [11] and choosing a threshold value, we use a feature selection technique to find the set of best features for a specific node. Using these, a SVM classifier is trained.

This allows much more accurate decisions than just setting a threshold on a single feature.

- Instead of a single split we perform ternary splits in order to be able to use very fast discriminators that only achieve a rough separation because of their simplicity. In fact, two thresholds are applied to the SVM output, based on a conservative exclusion principle. They are set so that the left branch excludes all classes belonging to the discriminator’s class set on the right and vice versa. The middle branch represents the “difficult” patterns as mentioned before (see fig. 1), which need more elaborate treatment.

To summarize the above we present the procedure for training a single node as pseudocode in algorithm 1. We use the notation  $\omega_k$  for a single class. The full set of classes is  $\Omega = \{\omega_k | k \in [0, K]\}$  where the classes 1..K refer to object classes and  $\omega_0$  is used for the background class.  $S_i$  denotes a set of classes which reaches node  $i$  and may be any subset of the full set  $S_i \subset \Omega$ . At the beginning the set of classes is initialized to the full set  $S_0 = \Omega$ .

For each node a set of candidate splits is evaluated which depends on the remaining classes at node  $i$ . Additionally, a state vector is maintained to keep track of the splits that have already been employed. For those particular split variants we have to use a more expensive discriminator with more features or eventually make a binary decision in order to avoid that we get trapped into an infinite recursion.

---

#### Algorithm 1 Expand Detector

---

```

Given Classes:  $S_i$ 
if [Remaining Number of Examples Below Threshold]
or [Maximum Tree Depth Reached] then
  - Return (Keeping Leaf Detector)
else
  - Expand Detector into Stump
  - Determine Candidate Splits for the Discriminator
    ( $S_{\text{left}} \Leftrightarrow S_{\text{right}}$ )
  for all Candidate Splits do
    - Train a Stump (Train Discriminator and the Three
      Detectors, Selecting the Best Features at Each
      Node)
    - Evaluate Stump’s Cost
  end for
  - Choose Best Stump
  - Update State of Processed Split Candidates
  - Expand Left Detector (Classes  $S_n \setminus S_{\text{right}}$ )
  - Expand Middle Detector (Classes  $S_i$ )
  - Expand Right Detector (Classes  $S_n \setminus S_{\text{left}}$ )
end if

```

---

### 3.3. Variants of the splitting scheme

The proposed algorithm is in fact very flexible regarding the types of trees it can produce. In every expansion we have the possibility to choose any combination of two mutually exclusive class subsets to be discriminated. Depending on the selected candidate, the resulting tree can take any shape. Yet, the generation can be steered by imposing restrictions on the set of split candidates. In practice we are even enforced to do so because the size of the candidate set grows exponentially with the number of classes. So we have to give additional rules about which classes to separate. Needless to say, we would like to avoid ruling out the formation of powerful trees. There are no clear guidelines for this however, and we therefore test three variants in the experimental section.

More specifically, our investigations concentrate on 3 families of split candidates. **Variante A** incorporates all possible splits of *pairs* of single classes (e.g.  $2 \Leftrightarrow 5$ ), with patterns of other classes going either way. **Variante B** consists of splits that involve all classes, inspired by the approach in [1, 2]. For example, given 5 classes, possible splits would be  $1, 3 \Leftrightarrow 2, 4, 5$  or  $1, 2, 4 \Leftrightarrow 3, 5$ . We call them *full splits* in the sequel. As the number of these candidates is immense, we made a random selection of split options, and then for each node chose the option with the lowest cost. **Variante C** is a special case in the sense that it produces a standard *cascade* classifier [12]. It exclusively uses “background-vs-all” splits (e.g.  $0 \Leftrightarrow 1, 2, 3, 4, 5$  in the case of 5 classes), varying only the number of features. These 3 variants are investigated in section 4 in order to find out which is the best strategy, in particular with respect to a growing number of classes.

### 3.4. Split candidate evaluation

For the split candidate evaluation we focus on the mean computation cost for the resulting stump. This cost depends on that of the discriminator and the distribution of the patterns over the detectors and their costs. Considering all possible patterns, there is a specific probability for each detector that it is reached. The mean cost is then the sum of the products of probability  $p_i$  and classifier cost  $c_{\text{classifier}_i}$  over all nodes  $i$ :

$$c_{\text{stump}} = c_{\text{discriminator}} + \sum_{\text{detectors: } i \in \{L, M, R\}} c_{\text{detector}_i} \cdot p_i \quad . \quad (1)$$

The cost of any of these 4 classifiers (one discriminator plus 3 detectors) can in turn be approximated by the product of the number of its features  $n_f$  and the number of Support Vectors  $n_{SV}$ :

$$c_{\text{classifier}} = n_f \cdot n_{SV} \quad . \quad (2)$$

### 3.5. Parameter choice for SVM training

SVM learning requires to choose a trade-off between training error and margin. This is typically represented by a scalar parameter  $C$  which multiplies the values of the slack variables. Higher values of  $C$  fit the separating hyperplane better to the training data. Lower values produce a smoother surface but allow more false classifications. The difficulty is to find a value which provides a good separation between positives and negatives but does not overfit the data. This is especially important as we want to set the thresholds to the difficult patterns tight but conservative. We set  $C$  automatically depending on the feature vectors  $x$  of the training set, namely to the expected value of the inverse of the inner product of them  $E[\frac{1}{x*x}]$ . This is the proposed default value in the SVM implementation that we used [6]. We found that it indeed yields a good compromise of accuracy and smoothness of the separation plane.

## 4. Experiments

### 4.1. Setup

In order to investigate our approach, we performed a number of experiments on a basic tree of depth 3. As mentioned in section 3.3 we have investigated different variants of the splitting scheme, namely pairwise splits (A), full splits (B) and a detector cascade (C). The experiments involved different configurations of classes in different configurations.

The database used for the experiments is the “MIT-CSAIL Database [10] of Objects and Scenes”. From this we considered up to 10 classes, namely “carFrontal”, “carSide”, “keyboard”, “light”, “firehydrant”, “trashWhole”, “mouse”, “mousepad”, “poster”, and “speaker”. These classes contain a varying amount of patterns ranging from 70 to about 500. The background collection was produced by random sampling of image frames from other scene images containing no instances of the object classes in question. For the experiments we used a set of 16000 items collected that way. Fig. 4 illustrates the contents of the database at the example of one scene image, with the objects to be detected marked as cutouts. It exhibits the complexity of the problem with the cluttered background producing an enormous and rich set of patterns to be evaluated. Also, it shows a selection of further object samples. It is worth to note the large intra-class variance of the object patterns, which makes the detection task even harder.

The classifiers used are SVM classifiers with a 2nd degree polynomial kernel. For the discrimination of class sets we used 32 and 64 features. For the final detector nodes 512 features were used.

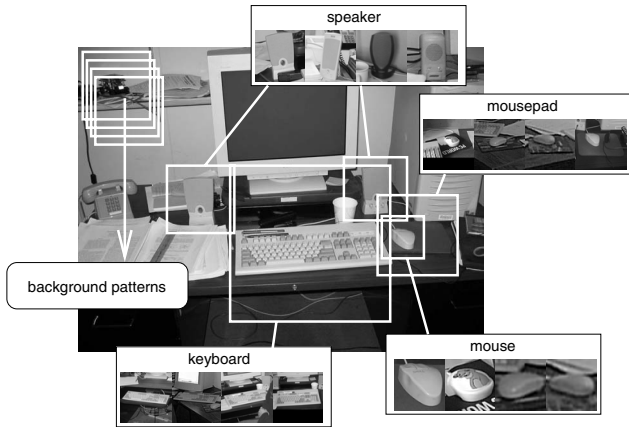


Figure 4. Examples of the image database.

An extended set of Haar Wavelet features at scales 2,3,4 was used as basic feature set. It consists of 1425 features in total. In order to reduce the number of features involved we applied two different feature selection techniques. For the discriminators with 32 and 64 features we employed an AdaBoost based technique. For the detection with 512 features we followed an SVM gradient based approach [4]. The choice of two different methods was made because the latter delivers a good overview of all features that are discriminative, but it does not take into account possible redundancies with other features. AdaBoost in contrast explicitly focuses on complementary features and gives much better results for small feature sets.

In order to set thresholds on the SVM output of the discriminators, we applied the following method. First, from the given distribution of a class set (left/right), a point is determined such that a high percentage of the objects are classified correctly. Then a safety margin is added based on the standard deviation  $\sigma$  of the distribution. This procedure has been chosen to increase the robustness against outliers in the distribution. Just setting the threshold to the exact border often produces overly optimistic or pessimistic values. The actual parameters were empirically determined as 99% for the ratio and  $0.75\sigma$  for the size of the margin.

## 4.2. Results

Fig. 5 shows the evaluation cost for all tree variants and for different sizes of class sets ranging from 2 to 10. It includes the ratio of the cost between variant C and A. The outcomes indicate that in the case of 2 and 3 classes the cost is quite similar for all variants. Furthermore, for an increasing number of classes the cost increases for all variants, which is to be expected. But an important aspect is that this increase behaves differently for the individual variants. For variants B and C the increase is much larger than

for variant A. Finally, with 10 classes, the cost for variant C has grown to about 4 times the cost of variant A. This is also expressed by the ratio curve which shows a decreasing tendency.

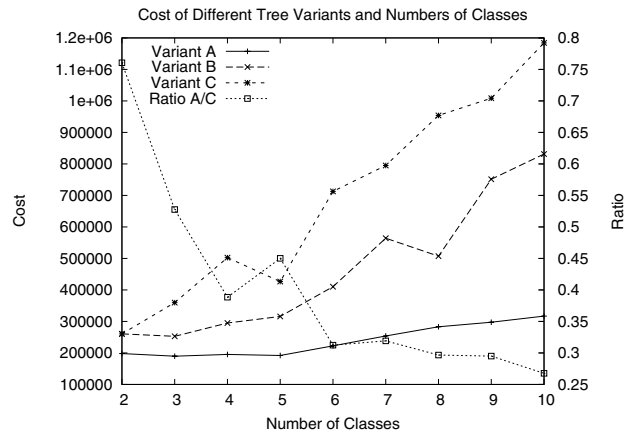


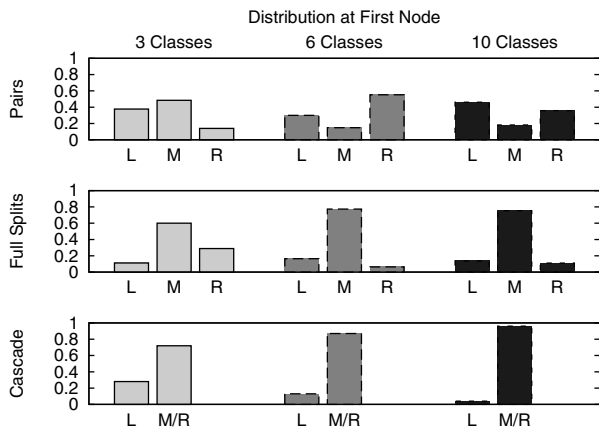
Figure 5. Development of the cost for different tree variants as a function of the number of classes. A: Single Pair Splits, B: Full Splits, C: Cascade.

An explanation for this outcome is found by investigating the distribution on the first tree node (see fig. 6). It shows that for 3 classes, the cascade is able to drop about one 3rd of the background. But for an increasing number of classes this ratio constantly gets smaller. For 10 classes it has already decreased to only 4%. Or stated the other way around: About 96% of all cases are still in the processing pipeline and have to be sent through further – more expensive – classifiers.

A similar behavior applies to variant B. In the case of 3 classes, the discriminator delivers a reliable decision for about 40% of all cases<sup>1</sup>. For the remaining 60%, we still have to take into account all possible hypotheses. With an increasing number of classes, the amount of difficult cases increases to about 90% for 10 classes.

For variant A, we typically observe a partitioning of the patterns among the branches. Usually there are not more than two 3rd concentrated in one branch. Clearly, the gain of information is relatively low for a single class pair split – especially when having many classes in total. But in addition to splitting two object classes it also partitions the background set. And that in turn is important because it is easier to detect objects from only a subset instead of all possible background items. As the results show, that effect plays a crucial role.

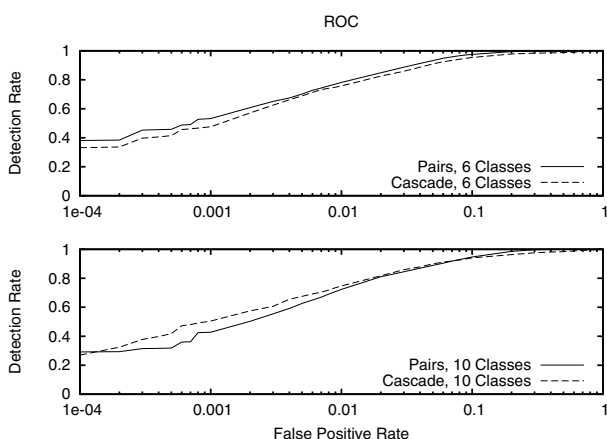
<sup>1</sup>Note that the test against background still has to be carried out. But as there is only a subset of classes to be considered and a subset of all background patterns we can expect a lower cost for this task.



**Figure 6. Distribution to subsequent branches at the first split for different tree variants and numbers of classes.**

It is important to note that Variants A and B have the important advantage over Variant C that they already subdivide patterns into smaller sets of classes, so that subsequent recognition of the separate classes has already progressed to a good extent.

Of course, it is not only the total cost which is important for a classifier, but also its accuracy. We investigated this aspect by producing ROC curves which provide an overview of the possible detection and false positive rates. Fig. 7 shows a comparison for a selection of different settings. The plot shows, that the ROC curves are typically very close to each other for a specific setting and therefore the accuracy is in a comparable range.



**Figure 7. Detection performance in terms of ROC curves for different tree variants and numbers of classes.**

## 5. Conclusion

We have presented a novel algorithm for growing a decision tree focusing on the fast subdivision of a given set classes into subsets that are distinguished jointly from the background. The algorithm optimizes the overall cost of the detector. We have investigated 3 different variants of the algorithm. For the resulting decision trees, cost and detection rates have been determined in order to find the best strategy in view of a growing number of classes. We have found that using pairwise splits provides the best results as the cost increase is much lower than for the other variants and the detection rates will be comparable.

Further work includes the development of smarter methods for class splits at the nodes and the introduction of variable tree depths.

## Acknowledgments

The authors would like to thank the IM2 project for supporting this work.

## References

- [1] Y. Amit, D. Geman, and X. Fan. A coarse-to-fine strategy for multi-class shape detection. *PAMI*, 26(12):1606–1621, 2005.
- [2] G. Blanchard and D. Geman. Hierarchical testing designs for pattern recognition. *Annals of Statistics*, 33(3):1155–1202, 2005.
- [3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall/CRC, 1984.
- [4] V. Depoortere, J. Cant, B. Van den Bosch, J. De Prins, R. Fransens, and L. Van Gool. Efficient pedestrian detection: a test case for svm based categorization. In *Proceedings Cognitive Vision Workshop*, 2002.
- [5] T. Evgeniou, M. Pontil, C. Papageorgiou, and T. Poggio. Image representation for object detection using kernel classifiers. In *ACCV*, pages 687–692, 2000.
- [6] T. Joachims. Making large-scale support vector machine learning practical. In A. S. B. Schölkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.
- [7] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1986.
- [8] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1993.
- [9] B. Schölkopf and A. J. Smola. *Learning with Kernels*. The MIT Press, 2001.
- [10] A. Torralba, K. P. Murphy, and W. T. Freeman. Mit-csail database of objects and scenes.
- [11] A. Torralba, K. P. Murphy, and W. T. Freeman. Sharing features: efficient boosting procedures for multiclass object detection. In *CVPR*, volume 2, pages 762–769, 2004.
- [12] P. Viola and M. Jones. Robust real-time object detection. In *Workshop on Statistical and Computational Theories of Vision – Modeling, Learning, Computing and Sampling*, 2001.