Making a Shallow Network Deep: Growing a Tree from Decision Regions of a Boosting Classifier

Tae-Kyun Kim*
http://mi.eng.cam.ac.uk/~tkk22
Ignas Budvytis*
ib255@cam.ac.uk
Roberto Cipolla
cipolla@cam.ac.uk

Department of Engineering
University of Cambridge
Trumpington Street, Cambridge
CB2 1PZ, UK
*indicates equal contribution.

Abstract

This paper presents a novel way to speed up the classification time of a boosting classifier. We make the shallow (flat) network deep (hierarchical) by growing a tree from the decision regions of a given boosting classifier. This provides many short paths for speeding up and preserves the reasonably smooth decision regions of the boosting classifier for good generalisation. We express the conversion as a Boolean optimisation problem, which has been previously studied for circuit design but limited to a small number of binary variables. In this work, a novel optimisation method is proposed for several tens of variables, i.e. weak-learners of a boosting classifier. The method is then used in a two stage cascade allowing the speed-up of a boosting classifier with any larger number of weak-learners. Experiments on the synthetic and face image data sets show that the obtained tree significantly speeds up both a standard boosting classifier and Fast-exit, a prior-art for fast boosting classification, at the same accuracy. The proposed method as a general meta-algorithm is also shown useful for a boosting cascade, since it speeds up individual stage classifiers by different gains. The proposed method is further demonstrated for rapid object tracking and segmentation problems.

1 Introduction

Boosting is a popular method in object detection [1], tracking [2] and segmentation [3] problems, which typically demand very fast classification. Boosting makes a decision by aggregating simple weak-learners e.g. Haar-like features, which are computed very fast on an integral image. Despite its efficiency, it is often required to further reduce the classification time. A cascade of boosting classifiers, which could be seen as a degenerate tree (see Figure 1(a)), effectively improves the classification speed: by filtering out majority of negative class samples in its early stages [1]. Designing a cascade, however, involves manual efforts for setting a number of parameters: the number of classifier stages, the number of weak-learners and the threshold per stage.

In this work, we propose a novel way to reduce down the classification time of a boosting classifier up to an order of magnitude without sacrificing its accuracy, not relying on a

BMVC 2010 doi:10.5244/C.24.57

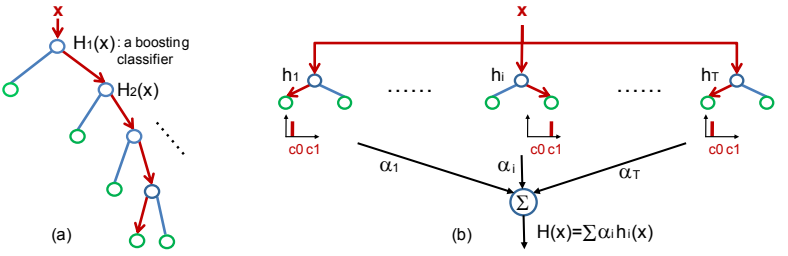


Figure 1: **Boosting as a tree.** *(a) A boosting cascade is seen as an imbalanced tree, where each node is a boosting classifier. (b) A boosting classifier has a very shallow and flat network where each node is a decision-stump i.e. weak-learner.*

design of cascade. The chance for improvement comes from the fact that a standard boosting classifier can be seen as a very shallow network, see Figure 1(b), where each weak-learner is a decision-stump and all weak-learners are used to make a decision. The flat structure ensures reasonably smooth decision regions for generalisation, however it is not optimal in classification time. The proposed method converts a shallow network (a boosting classifier as input) to a deep hierarchical structure (a decision tree as output). The obtained tree speeds up a boosting classifier by having many short paths: easy data points are classified by a small number of weak-learners. Since it preserves the same decision regions of the boosting classifier, the method alleviates a highly-overfit behaviour of conventional decision trees. We introduce a novel Boolean optimisation formulation and method. A boosting classifier splits a data space into 2^n primitive regions by n binary weak-learners. The decision regions of the boosting classifier are encoded by the boolean codes and class labels of the primitive regions. A decision tree is then grown using the region information gain. Further details are about a better way of packing the region information (Section 5.1) and the two stage cascade allowing the conversion with any number of weak-learners (Section 5.2). Without designing a many-stage cascade our method offers a convenient way of speeding up, while the method incorporated in such a cascade could provide a further speed-up.

The paper is organised as follows: Section 2 reviews related work. Overview of the proposed method is given in Section 3 and the formulation as Boolean optimisation in Section 4. Section 5 presents the proposed solution. Experimental results are shown in Section 6 and the conclusion is drawn in Section 7.

2 Related work

For speeding up the classification of a boosting classifier, the shortest set of weak-learners for a given error rate has been obtained by the sequential probability ratio test in the work of Sochman et al. [9]. It takes an early exit when the boosting sum reaches a certain value whose sign cannot be altered by the remaining weak-learners. Similarly, Zhou has proposed Fast exit method [11] (see Section 5.2 for details). This line of methods utilises so called *a single path of varying length*, while our tree method *multiple paths of different lengths*. The proposed method yields a more optimal speed (see Section 6).

The closest work to ours is Zhou's [11]. He has introduced representation of a boosting classifier by a Boolean table and implemented a binary decision tree [11]. His solution, however, is a brute force search for all possible tree configurations, which is highly

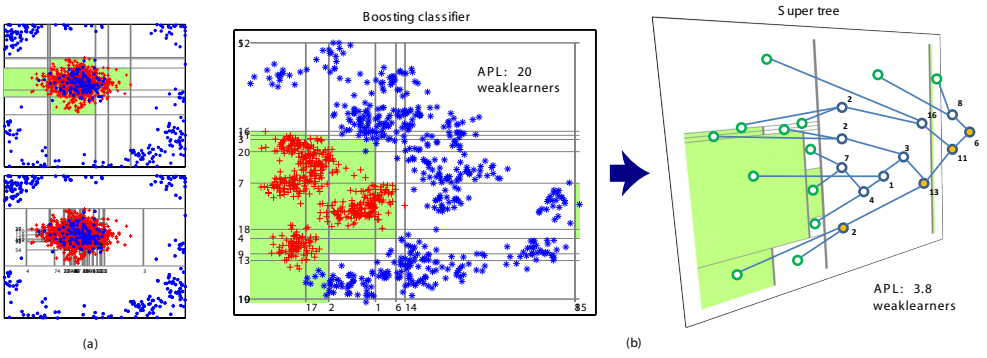


Figure 2: **Converting a boosting classifier into a tree for speeding up.** *(a) The decision regions of a boosting classifier (top) are smooth compared to a conventional decision tree (bottom). (b) The proposed conversion preserves the Boosting decision regions and has many short paths speeding up 5 times.*

computationally-costly. It therefore affords to only about 5 and 10 weak-learners. The speed gain reported was not significant over a standard boosting classifier and Fast exit method.

Tree-structured multiple boosting classifiers have been proposed for multi-pose or multi-category detection problems. The common structure is a tree hierarchy each path of which is a strong boosting classifier. Torralba et al. have proposed sharing weak-learners among multiple boosting classifiers [5] for accelerating classification speed. While Torralba's method requires pre-defined sub-category labels, the methods in [6, 7, 8] automatically learn the sub-category labels for multiple boosting classifiers in a tree. Whereas all these methods are useful for *multiple* boosting classifiers, our work focuses on a *single* boosting classifier. A further conceptual difference lies in that the previous studies [6, 7, 8, 12] present a novel way of learning boosting classifiers and ours takes a boosting classifier learnt in a standard way as input. We do not alter the decision regions of an input classifier but speed it up.

Boolean expression minimisation is to minimize the number of terms and binary variables in the Boolean expression. Algorithms for the minimisation have mainly been studied in the circuit design [15]. Since circuits have strictly predefined specifications, exact minimization was the goal of most studies. The complexity of a logic expression rises exponentionally when the number of binary variables increases. Therefore, conventional minimisation methods are limited to a small number of binary variables, typically from a few to about 15 variables [15]. Boolean minimisation has been also applied to size down a redundant decision tree, represented by a Boolean table [16].

3 Conversion of a boosting classifier into a tree

Both a boosting classifier and a decision tree are composed of weak-learners (or called decision-stumps/split-nodes). Whereas a boosting classifier places decision stumps in a flat structure, a decision tree has a deep and hierarchical structure (see Figure 1(b) and 2(b)). The different structures lead to different behaviours: Boosting has a better generalisation via reasonably smooth decision regions. See Figure 2(a) for the decision regions of the two methods. Here a part of negative (blue) data points are scattered in the middle of positive (red) samples. Whereas a conventional decision tree forms complex decision regions trying

classification of all training points, a boosting classifier exhibits a reasonable smoothness in decision regions. We propose a method to grow a tree from the decision regions of a boosting classifier. As shown in Figure 2(b), the tree obtained, called *super tree*, preserves the Boosting decision regions: it places a leaf node on every region that is important to form the identical decision boundary (i.e. accuracy). In the mean time, Super tree has many short paths that reduce the average number of weak-learners to use when classifying a data point. In the example, super tree on average needs 3.8 weak-learners to perform classification whereas the boosting classifier needs 20: all 20 weak-learners are used for every point.

4 Boolean optimisation formulation

A standard boosting classifier is typically represented by the weighted sum of binary weak-learners as

$$\mathrm{H}(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i \mathrm{h}_i(\mathbf{x}), \tag{1}$$

where α_i is the weight and h_i the i-th binary weak-learner in $\{-1, 1\}$. The boosting classifier splits a data space into 2^m primitive regions by m binary weak-learners. Regions $R_i, i = 1, \ldots, 2^m$ are expressed as boolean codes (i.e. each weak-learner h_i corresponds to a binary variable w_i). See Figure 3 for an example, where the boolean table is comprised of 2^3 regions. The region class label c is determined by Equation 1. Region R_8 in the example does not occupy the 2D input space and thus receives the *don't care* label marked "x" being ignored when representing decision regions. The region prior $p(R_i)$ is introduced for data distribution as $p(R_i) = M_i/M$ where M_i and M are the number of data points in the i-th region and in total. The decision regions of the boosting classifier are encoded by a set of regions represented as

$$\begin{cases} B(R_i): \text{ boolean expression} \\ c(R_i): \text{ region class label} \\ p(R_i): \text{ region prior} \end{cases} \tag{2}$$

With the region coding, an optimally short tree is defined in terms of average expected path length of data points as

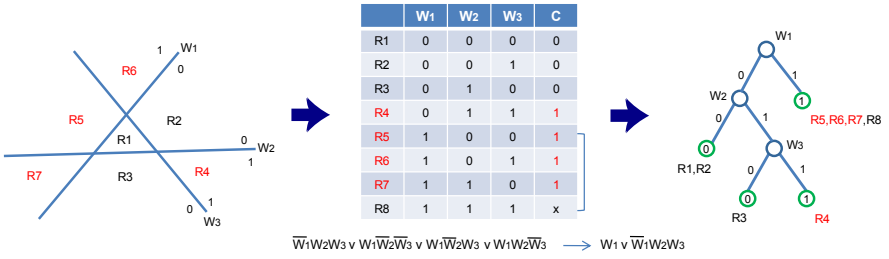
$$\mathbf{T}^* = \min_{\mathbf{T}} \sum_i E(l_{\mathbf{T}}(R_i)) p(R_i), \tag{3}$$

where $\mathbf{T}$ denotes all possible configurations of a decision tree. $E(l_{\mathbf{T}}(R_i))$ is the expected path length of the i-th region in $\mathbf{T}$. The path length is simply the number of weak-learners (or split-nodes) on the path to the i-th region. The decision tree should closely duplicate the decision regions of the boosting classifier as an optimisation constraint: the regions that do not share the same class label $c(R_i)$ must not be put in the same leaf-node of the tree. Any regions of *don't care* labels are allowed to be merged with other regions for the shortest path possible.

4.1 Discussion on boolean expression minimisation

The boolean expression for the table in Figure 3 can be minimised by optimally joining the regions that share the same class label or *don't care* label as

$$\begin{gathered} \overline{\mathrm{w}}_1 \mathrm{w}_2 \mathrm{w}_3 \vee \mathrm{w}_1 \overline{\mathrm{w}}_2 \overline{\mathrm{w}}_3 \vee \mathrm{w}_1 \overline{\mathrm{w}}_2 \mathrm{w}_3 \vee \mathrm{w}_1 \mathrm{w}_2 \overline{\mathrm{w}}_3 \\ \longrightarrow \mathrm{w}_1 \vee \overline{\mathrm{w}}_1 \mathrm{w}_2 \mathrm{w}_3 \end{gathered} \tag{4}$$



	W1	W2	W3	C
R1	0	0	0	0
R2	0	0	1	0
R3	0	1	0	0
R4	0	1	1	1
R5	1	0	0	1
R6	1	0	1	1
R7	1	1	0	1
R8	1	1	1	x

Figure 3: **Boolean expression minimisation for an optimally short tree.** *(a) A boosting classifier splits a space by binary weak learners (left). The regions are represented by the boolean table and the boolean expression is minimised (middle). An optimal short tree is built on the minimum expression (right).*

where $\vee$ denotes OR operator. The minimised expression has a smaller number of terms. Only the two terms, w_1 and $\overline{\mathrm{w}}_1\mathrm{w}_2\mathrm{w}_3$ are remained representing the joint regions $R_5 - R_8$ and R_4 respectively. A short tree is then built from the minimised boolean expression by placing more frequent variables at the top of the tree (see Figure 3(right)). The method for Boolean expression minimisation is close, but not suited to our problem that involves a large number of variables i.e. weak-learners. Furthermore, all regions are treated with equal importance in the kind of methods, while an optimally short tree is learnt by considering data distribution i.e. region prior in Equation 3.

5 Growing a super tree

We propose a novel boolean optimisation method for obtaining a reasonably short tree for a large number of weak-learners of a boosting classifier. The classifier information is efficiently packed by using the region coding and a tree is grown by maximising the region information gain. First, a base algorithm is explained, then its limitations and an improved method are presented. We use the notations in Section 4 to describe the algorithm.

Regions of data points. The number of primitive regions 2^m is intractable when m is large. Regions R_i that are occupied by any training data points are only taken as input s.t. $p(R_i) > 0$. The number of input regions is thus smaller than the number of data points. Regions with no data points are labeled *don't care*.

Tree growing by the region information gain. Huffman coding [7] is related to our optimisation. It minimises the weighted (by region prior in our problem) path length of code (region). The technique works by creating a binary tree of nodes by maximising the entropy-based information gain. We similarly grow a tree based on the region information gain for an optimally short tree. For a certain weak-learner $w_j, j = 1, ..., m$, the regions in the left split and the right split w.r.t. the weak-learner are readily given from the boolean expressions as

$$\begin{aligned} \mathbf{R}_l &= \{R_i | B(R_i) \wedge \overline{\mathrm{w}}_1 \cdots \mathrm{w}_j \cdots \overline{\mathrm{w}}_m) = 0\} \\ \mathbf{R}_r &= \mathbf{R}_n \setminus \mathbf{R}_l \end{aligned} \tag{5}$$

where $\mathbf{R}_n$ is the set of regions arriving at the node n and $\wedge$ is AND operator. At each node, it

is found the weak-learner that maximises

$$\Delta I = -\frac{\sum_{\mathbf{R}_l} p}{\sum_{\mathbf{R}_n} p} E(\mathbf{R}_l) - \frac{\sum_{\mathbf{R}_r} p}{\sum_{\mathbf{R}_n} p} E(\mathbf{R}_r) \tag{6}$$

where p is the region prior and E is the entropy function of the region class distribution, which is

$$Q(c^*) = \sum_{\mathbf{R}_c^*} p, \text{ where } \mathbf{R}_c^* = \{R_i | c(R_i) = c^*\}. \tag{7}$$

The node splitting is continued until all regions in a node have the coherent region label. The key idea in the method has two-folds: 1) growing a tree from the decision regions and 2) using the region prior (data distribution). Compared to conventional decision trees built on data points, the proposed tree is grown upon smooth decision regions guaranteeing good generalisation. Using the region prior helps getting an optimally short tree in the sense of average path length of data points.

5.1 Extended regions

The base algorithm in the previous section is useful in a low dimensional input space. Only encoding the regions of data points, however, does not reproduce the exactly same decision regions of a boosting classifier. Regions of no data points may be assigned different class labels from the original ones, since they are *don't cares* in the tree learning. When a test point falls into those regions, the boosting classifier and the tree would make different decisions. This degrades classification accuracy when data has a high dimension for a given number of training data. Regions along the decision boundary are important although they do not have an actual data point when training. Covering as much of the regions as possible ensures good performance. Adding up the primitive regions, however, becomes soon computationally prohibitive.

Extended regions. The region transformation is proposed to cover the regions in a fairly sufficient and yet computationally tractable manner. It takes each primitive region of data point (R_i) multiple times (see Figure 8(left)) and pushes it closer into the decision boundary by randomly flipping 1's to 0's (if the region class is positive) or 0's to 1's (if negative) until the boosting sum gets close to 0. See Figure 4 for an example. The extended region ER_i is then obtained by replacing all 0's in the boolean code of the pushed region with *don't care* variables s.t. $B(ER_i) = \mathrm{w}_1\mathrm{x}\mathrm{w}_3\mathrm{x}\mathrm{x}$. Each extended region thus contains many primitive regions of the same class label including the ones near to the decision boundary. Since the region space is big enough, it is unlikely to get identical extended regions or many regions with significant overlaps by the random drawing. The extended regions maintain the region class label $c(R_i)$ and prior $p(R_i)$.

Modified region information gain. When splitting nodes (Equation 5) an extended region can be placed in both left and right splits due to the existence of *don't care* variables. The repetition of same extended regions at different nodes does not hinder from duplicating the decision regions but increases the average tree length. To compensate the repetition, the information gain is modified as

$$\Delta J = \left(\frac{|\mathbf{R}_l| + |\mathbf{R}_r|}{|\mathbf{R}_n|} \right)^t \Delta I \tag{8}$$

	W1	W2	W3	W4	W5	Sum	C
Weight	1.0	0.8	0.7	0.5	0.2	3.2	
Region	1	0	1	1	0	1.2	1
Boundary region	1	0	1	0	0	0.2	1
Extended region	1	x	1	x	x	0.2-3.2	1

Figure 4: **Extended region coding.**

Algorithm: Growing a super tree

Input: a set of data point regions R or extended regions ER, encoded by $\{B, c, p\}$

Output: a decision tree

1.Start with a root node $n = 1$ containing the list of all regions $\mathbf{R}_n$.
2.For i=1,...,m
3. Spit the node: $(\mathbf{R}_l, \mathbf{R}_r) = \text{split}(\mathbf{R}_n, \text{w}_i)$ (by (5)).
4. Compute the gain: $\Delta I = \text{gain}(\mathbf{R}_l, \mathbf{R}_r)$ (by (6) or (8) for the extended region).
5.Find w_i^* that maximises the information gain.
6.If the gain is sufficient, save it as a split node. Else, save it as a leaf node.
7.Go to a child of split node and recurse the steps 2-6 setting $\mathbf{R}_n = \mathbf{R}_l$ or $\mathbf{R}_r$.

Figure 5: **Pseudocode of the algorithm**

where ΔI is the information gain in Equation 6, which takes a value in $[-\infty, 0]$. The first term equals to one for the primitive regions but is in the range of $[1, 2]$ for the extended regions. The modified gain penalises weak-learners that place many extended regions in both splits. The weight factor t is set empirically (see Figure 8). See Figure 5 for the pseudo-code.

5.2 Two stage cascade

The proposed method well scales up to several tens of weak-learners on a standard PC. For allowing any larger number of weak-learners of a boosting classifier, a two stage cascade is exploited. It places the super tree at the first stage and the fast-exit method at the second stage. Fast-exit yields exactly the same accuracy regardless of the number of weak-learners used, so the two stage cascade does. Fast-exit speeds up a boosting classifier by applying weak-learners in the order of weights α and exiting as soon as the boosting sum (Equation 1) reaches to the value whose sign cannot be altered by the remaining weak-learners. The proposed cascade significantly speeds up a cascade of the fast-exit at both stages as well as a standard two stage boosting cascade (see Section 6.2). The use of super tree is of course not limited to a two stage cascade. It guarantees a speed up over a multi-stage boosting cascade by replacing each stage of a boosting classifier with a Super Tree. Please refer to the speed gains obtained for different number of weak-learners of a single stage boosting classifier in Figure 7.

6 Experiments

6.1 Classification of synthetic 2D data

We have made twelve 2D synthetic data sets. Data points of two classes were generated from Gaussian mixtures as exemplified in Figure 6. The six test sets were created by randomly

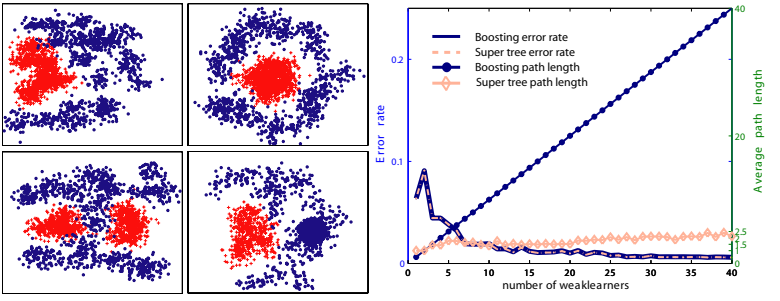


Figure 6: **Experimental results on the synthetic data.** *Examples of 2D synthetic data sets (left). Super tree obtains the same accuracy as the booting classifier significantly shortening the average path length (right).*

No. of weak learners	Boosting			Fast exit (cascade)			Super tree (cascade)		
	False positives	False negatives	Average path length	False positives	False negatives	Average path length	False positives	False negatives	Average path length
20	501	120	20	501	120	11.70	476	122	***7.51***
40	264	126	40	264	126	23.26	231	127	***12.23***
60	222	143	60	222	143	37.24	212	142	***14.38***
100	148	146	100	148 (144)	146 (149)	69.28 (37.4)	(145)	(152)	(***15.1***)
200	120	143	200	120 (146)	143 (148)	146.19 (38.1)	(128)	(146)	(***15.8***)

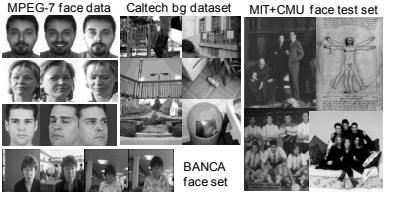


Figure 7: **Experimental results on the face images.** *Example face images are shown in right.*

perturbing the train sets. We have compared the two methods here: a boosting classifier (AnyBoost implementation [11]) and the proposed tree using the data point regions. Vertical and horizontal lines are weak-learners of boosting. Figure 6(right) shows the results. The left and right y-axis in the graph show the classification error rate and the average path length i.e. number of weak-learners used per point respectively. Note first that the both methods do drop the error rate when the number of weak-learners is increased indicating good generalisation. The proposed method exhibited the same accuracy as the boosting classifier for all number of weak-learners. While the boosting classifier linearly increased the average path length for the number of weak-learners, the proposed method quickly converged significantly reducing down the average path length. At 40 weak-learners, the super tree speeds up the boosting classifier by 16 times.

6.2 Object detection

For training, we used the MPEG-7 face data set that has 11,845 face images. BANCA face set (520 faces) and Caltech background image sets (900 images) were exploited for bootstrapping. The total number of negative images for training, which were either bootstrapped or randomly drawn, is 50,128. We used 21,780 Haar-like features on integral images as weak-learners. We have tested on the MIT+CMU frontal face test set which consists of 130 images with 507 labeled frontal faces. 507 face and 57000 random image patches were cropped and resized into 24x24 images. Example images are shown in Figure 7. The methods compared include a standard boosting classifier, Fast exit, Fast exit (two-stage cascade), Super tree and Super tree (two-stage cascade). For the super tree, we used the extended regions. The conversion time for the Super Tree for e.g. 40 weak-learners took about an hour.

No. weak-learners		10	20	30	40	50	60
No. per region		1	1	2	10	40	50
False+es/ False–es	super tree	593/157	367/146	292/136	262/129	203/142	224/129
	Boosting	588/157	378/143	291/137	264/126	202/142	222/143

Power	0.5	1	3	5	10
Avg path length	16.4	12.3	11.9	14.5	15.8
False +es/False -es	246/121	247/123	237/124	235/120	251/132

Figure 8: **Performance of super tree** *for the different numbers of extended regions per region (left) and for varying power in the information gain (right).*

Fixing the accuracy at 0 threshold, we have compared the average path lengths of the methods in Figure 7. The super tree speeds up the boosting classifier by 3-4.3 times and even the fast exit by 1.6-2.6 times. The two-stage cascade solution of 60 weak-learner super tree and 200 weak-learner fast exit outperformed the standard boosting by 6.6-12.7 times and even the two-stage cascade of 60 and 200 weak-learner fast exits by 2.5 times. Note that the super tree exploits various combinations of weak-learners (i.e. paths) for an optimal classification speed, whereas the fast exit takes the combinations always in the order of the weak-learner weights.

Figure 8 shows performance of the super tree for the two internal parameters: the number of extended regions per primitive region and the power in the information gain (Equation 8). To obtain the close accuracy to the boosting classifier, the required number of extended regions per region grew as the number of weak-learners of Boosting increased. For about the given number of training samples, using 200 extended regions and 100 weak-learners would start hitting theoretical memory boundaries. As shown in right, the performance is not very sensitive to different power values in the range. The number of weak-learners and extended regions was set as 40. Power 1-5 gave the best performance. The values smaller than 0.5 increased the average path length and the values larger than 10 increased the error rate.

Super tree vs a conventional decision tree. Single conventional decision trees of all possible pruning [14] were very poor. The best accuracy of the conventional tree (false positives: 1995/false negatives: 120) is by far worse than that of the super tree of 20 weak-learners (false positives: 476/false negatives: 122). The super tree was even shorter than the decision tree: the depth of the super tree and conventional tree was about 7.5 and 9 respectively.

Tracking and Segmentation. We have demonstrated the usefulness of super tree for rapid object tracking and image semantic segmentation problems. The super tree exhibited better tracking performance than a boosting classifier by the benefits in the execution time of the tracker. It also showed a significant speed-up at the same accuracy over a boosting classifier for the segmentation by pixel classification. Refer to [19] for more details.

7 Conclusion

We have proposed a novel way to speed up a boosting classifier. The problem is formularised as boolean optimisation and a new optimisation method is proposed for a large number of weak-learners. The tree grown from the decision regions of a boosting classifier, called Super tree, provides many short paths and preserves the Boosting decision regions. The single super tree delivers the close accuracy to a boosting classifier with a great speed-up for up to several tens of weak-learners. The proposed two stage cascade allows any number of weak-learners. Experiments have shown that the tree obtained is reasonably short in terms of average path length outperforming a standard boosting classifier, fast exit, their cascade. The

method has been also demonstrated for rapid object tracking and segmentation problems.

References

[1] P. Viola and M. Jones, Robust real-time object detection, *Int'l J. Computer Vision*, 57(2):137–154, 2002.

[2] H. Grabner and H. Bischof, On-line boosting and vision, *Proc. IEEE Conf. CVPR*, pages 260–267, 2006.

[3] S. Avidan, SpatialBoost: Adding Spatial Reasoning to AdaBoost, *Proc. ECCV*, Graz, Austria, 2006.

[4] S.Z. Li and Z. Zhang, Floatboost learning and statistical face detection, *IEEE Trans. on PAMI*, 26(9):1112–1123, 2004.

[5] A. Torralba, K. P. Murphy and W. T. Freeman, Sharing visual features for multiclass and multiview object detection, *IEEE Trans. on PAMI*, 29(5):854–869, 2007.

[6] B. Wu and R. Nevatia, Cluster Boosted Tree Classifier for Multi-View, Multi-Pose Object Detection, *Proc. ICCV*, 2007.

[7] C. Huang, H. Ai, Y. Li, and S. Lao, Vector Boosting for Rotation Invariant Multi-View Face Detection. *Proc. ICCV*, 2005.

[8] Z. Tu, Probabilistic Boosting-Tree: Learning Discriminative Models for Classification, Recognition, and Clustering, *Proc. ICCV*, 2005.

[9] J. Sochman and J. Matas, WaldBoost Learning for Time Constrained Sequential Detection, *Proc. CVPR*, San Diego, USA, 2005.

[10] L. Mason, J. Baxter, P. Bartlett and M. Frean, Boosting algorithms as gradient descent, *Proc. Advances in Neural Information Processing Systems*, pages 512–518, 2000.

[11] S. Zhou, A binary decision tree implementation of a boosted strong classifier, *IEEE Workshop on Analysis and Modeling of Faces and Gestures*, pages 198–212, 2005.

[12] E. Grossmann, AdaTree: boosting a weak classifier into a decision tree, *IEEE Workshop on Learning in Computer Vision and Pattern Recognition*, pages 105–105, 2004.

[13] L. Breiman, Random forests, *Machine Learning*, 45:5–32, 2001.

[14] J. Quinlan, Bagging, boosting, and c4.5, *Proc. National. Conf. on Artificial Intelligence*, pages 725–730, 1996.

[15] H. Schwender, Minimization of Boolean Expressions Using Matrix Algebra, Technical report, Collaborative Research Center SFB 475, University of Dortmund, 2007.

[16] J. Chen, Application of Boolean expression minimization to learning via hierarchical generalization, *Proc. ACM symposium on Applied computing*, pages 303–307, 1994.

[17] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press and McGraw-Hill, 2001.

[18] G. Brostow, J. Shotton, J. Fauqueur and R. Cipolla, Segmentation and Recognition using Structure from Motion Point Clouds, *Proc. ECCV*, Marseilles, 2008.

[19] T-K. Kim, I. Budvytis, R. Cipolla, Making a Shallow Network Deep: Growing a Tree from Decision Regions of a Boosting Classifier, Techincal report, CUED/F-INFENG/TR633, Dept. of Engineering, Univ. of Cambridge, June 2009.