TCAS: A multiclass object detector for robot and computer vision applications

Rodrigo Verschae, Javier Ruiz-del-Solar

Advanced Mining Technology Center, Universidad de Chile, Chile

Abstract. Building efficient object detection systems is an important goal of computer and robot vision. If several object types are to be detected, the most simple solution is to run several object-specific classifiers independently of each other (in parallel). This solution is computationally expensive if several object classes are to be detected. In this paper, *TCAS*, a new classifier structure designed to be used on multiclass object detection problems is introduced as an alternative solution. TCAS offers an efficient solution and reduces the aggregated false detection rate. TCAS extends cascade classifiers (introduced by Viola & Jones) to the multiclass case and corresponds to a nested coarse-to-fine tree of multiclass nested boosted cascades. Results for three different object detection problems are presented: face and hand detection, robot detection, and multiview face detection. In the experiments, the obtained TCAS have classification times about 2-times shorter than the ones obtained using parallel cascades, and have the same or lower number of false positives (for the same detection rate).

1 Introduction

The development of robust and real-time object detection systems is an important goal of the robot-vision and computer-vision communities. Applications of multiclass detection systems include human-robot interaction (e.g. face and hand detection), and autonomous drivings system (e.g. car and sign detection), among many others.

For detecting all instances of an object-class, the sliding window approach [1] requires to perform an exhaustive search over image patches at different scales and positions. The classification of all possible windows (patches) may require a high computational power. To achieve an efficient detection, less processing time should be spent on non-object windows than on object windows. This can be achieved using cascade classifiers [1] (Fig. 1 (a)). Starting from the seminal work of Viola & Jones [1] on cascade boosted classifiers for object detection, several improvements have been proposed, with the introduction of nested cascades [2] being one of the most important ones.

In order to detected multiple classes of objects, the most simple solution is to use several cascades in a a concurrent manner, with each cascade trained for each particular class independently of each other. The main problem with this approach is that the processing time and the false positive rate (*fpr*) aggreegates with the number of classes. As a solution to these problems we propose a multiclass classifier with a tree structure. The proposed detector corresponds to a *multiclass* nested tree of multiclass nested cascades that compared to the use of parallel cascades presents a considerable gain in processing time. Results in 3 different detection problems are presented later.

TCAS learns an implicit partition of the target space, and performs an efficient coarse-to-fine (CTF) search in the object space, both, within each node of the tree, and while traversing it. On the contrary a cascade classifier learns a implicit partition of the target space that only allows to discriminate between the object class and non-object class. A *TCAS* is efficient because (1) it performs a CTF search, and (2) the feature evaluations and classifiers evaluations are shared between subsets of classes , reducing the computational complexity. The proposed *TCAS* structure (See Fig 1 (c) for an example) has similarities and differences with existing classifiers: Width-First-Search (WFS) trees [4], Cluster-Boosted-Trees (CBT) [5], and Alternating Decision Trees (ADT) [6].

A CBT has a similar structure to the a *TCAS*, but it is designed only for multiview detection problems. The main differences of CBT with *TCAS* are: (1) during training of a CBT, a partition of the object class is learnt, (2) the depth of CBT is the same for all branches, and (3) each node of the CBT contains a single "layer", while in a *TCAS* it contains a multiclass cascade. These differences reflect in that the *TCAS* performs a CTF search in the object space within each node of the tree, and the depth of each branch of the tree is variable.

In [4] a Width-First-Search (WFS) tree is proposed. The WFS tree structure is similar to the *TCAS*, but the main differences are that: (1) the WFS-tree is not nested, and (2) the WFS tree does not use CTF multiclass cascades. The main similarities are the use of Vectorboost and *WFS* search. The term WFS refers to the fact that at each node of the tree, any number of siblings can be followed while traversing the tree.

ADTs [6] are decision trees where each node does not only gives an output, but also a confidence, which is accumulated over all nodes that are visited. The main similarity between ADT and *TCAS* is that in an ADT, a leaf's output depends on all nodes of the branch the leafs belongs to (i.e. is "nested"), but it may also depend on nodes of other branches. This requires, in the case of the ADT, to maintain "global" weights during training, while in *TCAS* the weights are "local" to the node being trained.

In the following, background in presented in Section 2, followed by the proposed *TCAS* classifier (Section 3), an evaluation (Section 4), to finally conclude in Section 5.

2 Background

In [1] Viola and Jones proposed using cascade classifiers (Fig. 1 (a)) for the efficient detection of faces. Besides using rectangular features that can be efficiently evaluated, in the proposed cascade classifier the cascade's layers have an increasing computational complexity. Thus, because that most analized windows of any image do not contain faces, it can quickly reject windows that do not resemble faces. In [2] a nested cascade which reuses the confidence information from one layer in the next one was proposed. This nested structure is possible thanks of the additivity of the boosting model.

Multiclass Boosted Classifiers In [7] nested cascades were extended to the multiclass case, with each layer corresponding to a multiclass classifier and having a vectorised [4] form (see Fig. 1 (b)): $\boldsymbol{H}(x) = \sum_{t=0}^{T} \boldsymbol{h}_t(f_t(x))$. The classifier $\boldsymbol{H}(x)$ is trained using Vectorboost [4], which is a multiclass extension of Adaboost that assigns to each training example x_i an objective region in a vector space. An objective region is defined as the intersection of a set of half spaces, with each half-space defined by a vector, a, and a set of half-spaces defined by a parameter set $\mathbf{R}$. Under this setting, a sample x, belonging to class Y_q and represented by a parameter set $\mathbf{R}_q$, is classified correctly iff:

$$\forall \boldsymbol{a} \in \mathbf{R}_q, \langle \boldsymbol{a}, \boldsymbol{H}(x) \rangle \geq 0, \tag{1}$$

thus a class Y_q is assigned to a new sample x, if all the inequalities of R_q are fulfilled.

The weak classifiers $\boldsymbol{h}_t(f_t(x))$ are designed after the *domain-partitioning weak hypotheses* paradigm [8]: each feature domain $\mathbb{F}$ is partitioned into disjoint blocks $F_1, \ldots, F_J$, and a weak classifier $h(f(x), m)$ has a constant output for each partition block of its associated feature f. The weak classifier's components $\{h(f(x), m)\}_m$ can have different relations with each other (see [7] for details). In the present work we consider 2 cases: *independent* components and *coupled* components. For efficiency, weak classifiers are stored in look-up-tables (LUTs), which can be evaluated in constant time.

Multiclass Coarse-To-Fine Nested Cascades A coarse-to-fine (CTF) nested multiclass cascade [9] is used as a *TCAS* classifier's building block (see Section 3). A CTF nested multiclass cascade performs a search in the object target space, and this allows to reduce the processing time and to increase the accuracy of the cascade classifier. To perform the CTF search in the object target space, an *active mask* that has a binary output and that represents a subsets of active components at the current layer of the multiclass cascade is used. At layer k of the multiclass cascade, the *active mask* $\mathbf{A}_k(\cdot)$, is defined component-wise by: $\mathbf{A}_k(x, m) = u\left(H_k(x, m)\right) \prod_{i=0}^{k-1} \mathbf{A}_i(x, m)$, with $u(x)$ the unit step function. Note that if $\mathbf{A}_k(x, m)$ is 0, then $\mathbf{A}_j(x, m)$ is 0 for all $j \geq k$. Using the *active mask*, the output of a layer, k, of the cascade is:

$$\boldsymbol{H}_k(x) = \left[\boldsymbol{H}_{k-1}(x) + \sum_{t=1}^{T_k} \boldsymbol{h}_{k,t}(f_{k,t}(x))\right] \odot \mathbf{A}_{k-1}(x) \tag{2}$$

with, $\boldsymbol{H}_0(x) = \mathbf{0}$, $\mathbf{A}_0(x) = \mathbf{1}$, and with $\odot$ the point-wise product between two vectors. Eq.2 can be interpreted as verifying the condition of the input belonging to a particular class (Eq.1) at each layer of the cascade in a per component manner, and only for the subset of hypotheses that was already verified at the previous layers of the cascade.

Only non-zero components of $\mathbf{A}_{k-1}(x)$ need to be evaluated in Eq. 2 (at layer k). These non-zero components represent a subset of classes with positive output at the current layer (and potentially a positive output in the cascade). In this way, as a sample moves through the cascade, the output goes from a coarse output in the target space, to a finner one. Also, as in a standard cascade classifier, there is a CTF search on the non-object space: at each layer a window being analysed can be discarded.

3 *TCAS*: Nested Coarse-To-Fine Tree of Nested Cascades

The CTF multiclass cascade just described has the problem that if a subset of the classes to be detected is very "different" to another subset, it is not possible to train an efficient classifier, as too many weak classifier may need to be added to the cascade's layers.

The reason is that at each level of the system there are two competing goals: to discriminate between the background (the non-objects) and the objects, and also to discriminate among object classes. At the first levels of the tree, to discriminate between the objects and the non-objects is not difficult, but at later levels of the cascade, the non-objects to be discarded by the cascade resemble more the objects, thus features and weak classifiers that are more "specialized" to each class are needed. More precisely, this means that it becomes difficult to find features that can be used to discriminate between all objects classes from "all" background(s). The *TCAS* allows to partition the target space in a CTF way, in particular in later tree levels, but it also allows to share feature evaluations (whenever possible). Thus a trade-off between accuracy and efficiency is achieved.

A *TCAS* classifier, $\mathbb{T}$, corresponds to a directed tree, with each node having a variable number of siblings. A node $\mathbb{N}$ of $\mathbb{T}$ consists of: (a) a multiclass nested cascade classifier $\boldsymbol{H}^C_{\mathbb{N}}$ (as in Eq. 2), (b) a mask $\mathbf{A}_{\mathbb{N}} \in \{0,1\}^M$, (c) a "pointer", $p_{\mathbb{N}}$, to its direct ancestor, and (d) $n_{\mathbb{N}}$ siblings, $\{\mathbb{N}_s\}_{s=\{1,\ldots n_{\mathbb{N}}\}}$. Every node, $\mathbb{N}$, has a nested structure (depends on the output of its ancestors): $\boldsymbol{H}_{\mathbb{N}}(x) = \boldsymbol{H}_{p_{\mathbb{N}}}(x) \odot \mathbf{A}_{\mathbb{N}} + \boldsymbol{H}^C_{\mathbb{N}}(x)$, with $\boldsymbol{H}_{p_{\mathbb{N}}}(x)$ the output of the ancestor of $\mathbb{N}$, $p_{\mathbb{N}}$. Note that if $p_{\mathbb{N}}$ is the root of the tree then $\boldsymbol{H}_{p_{\mathbb{N}}}(\cdot) = \mathbf{0}$. Only non-zero components of $\mathbf{A}_{\mathbb{N}}$ need to be evaluated in $\boldsymbol{H}^C_{\mathbb{N}}(x)$, i.e. the CTF evaluation in the object target space is also used here, which further helps to achieve an efficient evaluation. The mask $\mathbf{A}_{\mathbb{N}}$ indicates which classifier's components are active at the current node and should be evaluated.

During training the following restriction is made (recall that $\mathbf{A} \in \{0,1\}^M$):

$$\mathbf{A}_{\mathbb{N}} = \sum_{s=Siblings(\mathbb{N})} \mathbf{A}_s \tag{3}$$

This means that the (binary) masks of any two siblings of node $\mathbb{N}$, $\mathbb{N}_i$ and $\mathbb{N}_j$, with $i \neq j$, holds that $\mathbf{A}_{\mathbb{N}_i} \odot \mathbf{A}_{\mathbb{N}_j} = 0$. This restriction implies an efficient recursive implementation, and simplifies the training process, because nodes at different branches of the tree do not depend on each other. Thanks to the structure of the tree, the output of a *TCAS*, $\mathbb{T}$, can be defined as the sum of the output all of its leafs $\mathbb{N}_{(1)}, \ldots, \mathbb{N}_{(n_{leafs})}$:

$$\boldsymbol{H}_{\mathbb{T}}(x) = \sum_{j=1,\ldots,n_{leafs}} \boldsymbol{H}_{\mathbb{N}_{(j)}}(x) \tag{4}$$

Eq. 4 can be implemented efficiently (see Alg.1) thanks to Eq. 3.

Tree Classifier Training The training of a *TCAS* (see Alg.2) is a top-down procedure, that recursively adds nodes to the tree starting from the root. Nodes are added by taking into account the nested multiclass cascade built by traversing the nodes' ancestors (line 3, using algorithm FLATTENING), and the training examples that have a target region defined by the active mask of the current node (lines 4 - 6). A node is trained and added (line 6, using algorithm TRAINNODE [9]) using the corresponding training examples and cascade. Later the components of a node are grouped (line 8), and the new siblings are initialised and added to the stack of nodes to be trained (lines 9 - 11). The training of the tree stops when the stack is empty (line 3). One reason for stopping adding nodes to a branch of the tree is that branch of the tree already fulfils the required false positive rate (*fpr*) (line 7). This means that the depth of the tree's branches is variable, which allows to adjust the complexity of each branch of the tree to the corresponding class.

The FLATTENING algorithm constructs a cascade associated to a tree's branch. Starting from a leaf node, it goes backward, up to the tree's root, taking only the components that are active at the leaf node. In the algorithm GROUPCOMPONENTS, the active components of a node are grouped and later each of these groups is assigned to a sibling. This procedure must fulfill two conditions: (1) the groups must have an empty intersection (Eq. (3)), and (2) the union of the groups must be equal to the input set. The 2nd condition indicates that if the input mask has only one active component, it must return the empty set (stop adding nodes to that branch) because the node does not have any sibling. In the present work the grouping is not done automatically, but it is preset.

4 Experimental Results

We present experimental results on three different detection problems: hand and face detection, multiview face detection, and robot detection. Each experiment seeks to analyse a different aspect of *TCAS*. In the first experiment (Face and Hand detection) we consider a multiclass detection problem and compare the use of several cascades versus the use of a *TCAS*. In the second experiment (multiview face detection) we analyse how *TCAS* scales up when increasing the number of classes. In the third experiment (robot detection) results on an multiview and multiclass setting showing the performance of *TCAS* when handling very different classes (humanoid and AIBO robots) are presented.

As in [9], in all following experiments, rectangular features are used in the first two levels of *TCAS* and modified LBP features ares used in later ones. Other parameter values are: *fpr* per layer $= 0.35$, *tpr* per layer per class $= 0.9995$, and target *fpr* $= 10^{-6}$.

Experiment 1: face and hand detection. We evaluate *TCAS* in a multiclass problem: hand and face detection. More specifically, three classes under frontal views are considered: faces, left fists and right fists ("frontal" indicates low in-plane rotations, up to $\pm \sim 15°$). Given that there is no standard datasets for this problem, we only present ROC curves for the detection of frontal faces in a the BioID database (1,521 images and 1,521 faces, DB that does not contains any hand), which gives and idea how well a specific 1-class detector (face detector) compares to a 3-classes detector (hands and faces) in detecting a specific class (faces in this case). The classifiers were trained using 4,900 examples per class (left hand fist, faces and right hand fist). In all trained classifier, all used parameters and (initial) training sets were exactly the same.

Table 1 presents the built detectors and their corresponding processing times. Note that it was not possible to obtain two-classes nor three-classes multiclass cascades [9], because the training procedure did not converge (even after adding a large number of weak classifiers during boosting, the *fpr* did not decrease and the detection rate did not increase, thus the required accuracy was not achieved). On the contrary the *TCAS* classifiers were built for the two-classes and three-classes problems as well as one-class cascade classifiers. In Fig.2 (a) the obtained detection rates of a *TCAS* (trained to detect all three classes but used to detect only faces), compared to the use of a one-class cascade trained specifically for face are presented. It can be observed that the accuracy of the *TCAS* is slightly higher for all operation points, being up to 1 percentage point higher for 5 FP. Moreover, in Table 1 can be observed that *TCAS*, compared to the use of parallel cascades, has a gain in processing time close to 1.6 times, i.e., in this problem *TCAS* is both faster and has a better accuracy than parallel cascades.

Experiment 2: number of classes and multiview face detection. We analyse the use of *TCAS* when handling an increasing number of classes. For this analysis we consider the multiview face detection problem with in-plane-rotations (RIP) in the range $[0, 360]$. This range was divided on non-overlapping view-ranges of 18°, and each range is defined an object class. The base range $[-45, 45)$ is divided in to the following 5 ranges: $\{[-45, -27] + i * 18\}_{i=0,\ldots,4}$. To cover the full range $[0, 360)$, classifiers are applied on rotated versions (0, 90, 180, and 270°) of the input or trained on classes that are defined by rotated versions of the 5 ranges. Different classifiers are trained to detected subsets of these classes: e.g in one case, twenty 1-class cascades are used in parallel, while in another case four 5-classes *TCAS* are used in parallel, and both systems are compared in a 20-classes problem. The CMU Rotated set [10], which consists of 50 images and 223 faces with in-plane rotations in [0,360], is used as test set.

We consider 4 cases, each one corresponding to a fixed number of classes ($n = 1, 5, 10, 20$) when training the classifiers. For the case $n = 1$, five 1-class cascades are trained, and they are applied 4 times (4 rotated images), which gives 20 cascades in total. Also, four 5-classes *TCAS* (applied to the image rotated in 0, 90, 180 and 270°), two 10-classes *TCAS* (applied to the image rotated in 0 and 180°), and one 20-classes *TCAS* (covering the whole range) are evaluated.

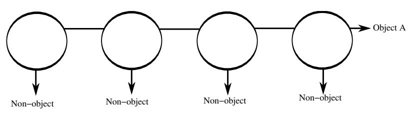
The training set for each of these ranges was built using the ISL WebImage dataset[1], with 800 labelled faces that were rotated randomly (uniform distribution) in the corresponding range. Also random translations and scale changes were applied, generating 1600 training samples per range. Per class, 3,200 negative examples were collected, i.e. when building a classifier for 5 objects classes, $16,000$ $(5 * 3,200)$ negative examples and $8,000$ $(5 * 1,600)$ positive examples are used. In the case of the 20-classes *TCAS* where 3,000 negative examples per class were used, because of memory restrictions. The maximum number of weak classifiers per node was set to $T_{max} = 50$; this triggers stopping the training of a node and starting the training the next tree's level).

The obtained processing times are presented in Table 2. The detection results are omitted for space reasons, but it can be stated that: the 5-classes *TCAS* and the 10-classes *TCAS* have a better performance than 20 parallel cascades, and a slightly worst performance is obtained by the 20-classes *TCAS*.

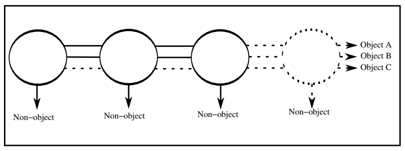
In terms of processing time, using four 5-classes *TCAS* is approximately 1.7 times faster than running 20 parallel cascades, using two 10-classes *TCAS* is 1.8 times faster than running parallel cascades, and the 20-classes *TCAS* is 1.2 times faster than running parallel cascades. The lower performance of the 20-classes *TCAS* may be due to the use of a slightly smaller training set, or it could be also due to a structural issue. Fig. 3 (a) shows a multiview detection example using the 10-classes TCAS classifier.

The training time grows quadratically with the number of classes, but if the size of training set is taken into account, it grows only linearly ($R^2 = 0.98$). This shows that the training procedure can scale up with the number of classes, and the main variable defining the training time is the number of training examples. The number of weak classifiers

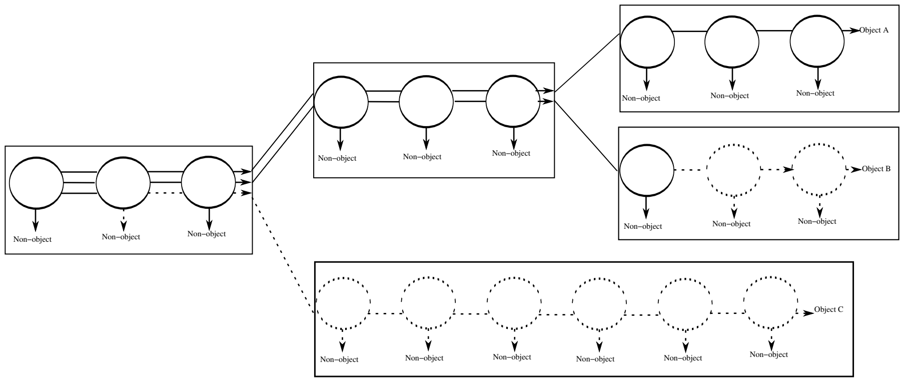
[1] Available at http://www.ecse.rpi.edu/∼cvrl/database/database.html



(a) Cascade (1-class)



(b) Multiclass Cascade (3-classes)



(c) TCAS (3-classes)

Fig. 1: Classifier's Structures. In (b) and (c) dashed lines show examples of branches that are not evaluated when performing a classification of a particular input.

Algorithm 1 $\text{EVALTREE}(\mathbb{T}, \mathbf{A}_0, x)$

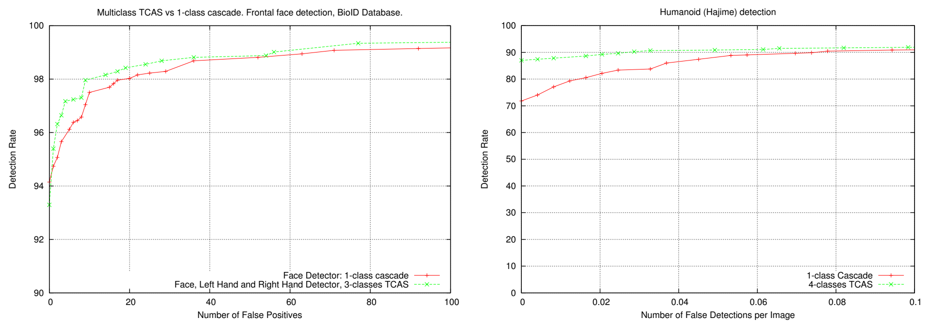
```
1:  $\boldsymbol{O}(x) \leftarrow \mathbf{0}$; $Push\,(ST, [\mathbb{T}, \mathbf{A}_0 \odot A_{\mathbb{T}}])$ // Initialize output and stack
2:  while $([\mathbb{N}, \mathbf{A}] \leftarrow Pop(ST))$ do
3:      $\boldsymbol{O}(x) \leftarrow \boldsymbol{O}(x) + \mathbf{A} \odot \boldsymbol{H}_{\mathbb{N}}^{C}$
4:      $\mathbf{A}_t \leftarrow \mathbf{A} \odot Binary(\boldsymbol{H}(x))$
5:      if $\mathbf{A}_t \neq 0$ then
6:          for each $\mathbb{N}_s \in Siblings(\mathbb{N})$ do
7:              if $\mathbf{A}_t \odot A_{\mathbb{N}_s} \neq 0$ then
8:                  $Push\,(ST, [\mathbb{N}_s, \mathbf{A}_t \odot Mask(\mathbb{N}_s)])$
9:              end if
10:         end for
11:     end if
12: end while
Output: $\boldsymbol{O}(x)$ // Only non-zero components of $\mathbf{A}$ need to be evaluated in $\boldsymbol{H}_t(x)$ at line 5.
13: $Binary(\boldsymbol{V})$ returns a vector which is, per component $m$: $(Binary(\boldsymbol{V}))_m = u\left((\boldsymbol{V})_m\right)$
```

Table 1: Processing times of frontal left and right hand, and frontal faces detectors.

Classifier	Classes	Time [sec]
Three 1-class cascades	Face & Hands	1.70
1-class + 2-class *TCAS*	Face & (Left & Right Hand)	1.27
1-class + 2-class *TCAS*	Left Hand & (Right Hand & Face)	1.39
3-class *TCAS*	Face & Hands	**1.01**

Algorithm 2 TRAINTREE(S, F_C, D, PV, NV)

```
1: $\mathbb{T} \leftarrow \{\emptyset\}$; $Push(ST, [\mathbb{T}, \mathbf{1}])$; $S = \{(x_i, \boldsymbol{a}_i)\}_{i=1,\ldots,n}$: Set of positive examples
2: while ($[\mathbb{N}, \mathbf{A}] \leftarrow Pop(ST)$) do
3:     $\boldsymbol{H}_t \leftarrow$ FLATTENING$(\mathbb{N})$
4:     $S_t \leftarrow filterExamples(\mathbf{A}, S)$
5:     $PV_t, NV_t \leftarrow filterExamples(\mathbf{A}, PV), filterExamples(\mathbf{A}, NV)$
6:     $\boldsymbol{H}_{\mathbb{N}} \leftarrow$ TRAINNODE$(\boldsymbol{H}_t, \mathbf{A}, S_t, F_C, PV_t, NV_t)$
7:     if $fpr$ $(\boldsymbol{H}_t + \boldsymbol{H}_{\mathbb{N}} \odot \mathbf{A}) \geq F_C$ then
8:         $\mathbb{G} \leftarrow GroupComponents(\mathbf{A})$
9:         for each $\mathbf{g} \in \mathbb{G}$ do
10:            $Push(ST, [newSibling(\mathbb{N}), \mathbf{g}])$
11:        end for
12:    end if
13: end while
Output: $\mathbb{T}$
14: $newSibling(\mathbb{N})$ creates an empty node and adds an edge (new sibling) to $\mathbb{N}$.
15: $F_C$: global false positive rate; $PV$ / $NV$: Positive/Negative Validation Set
16: $S_t \leftarrow filterExamples(\mathbf{A}, S)$: returns $S_t = \{(x_i, \boldsymbol{a}_i) \in S \mid (\boldsymbol{a_i} \odot \mathbf{A}) \neq \mathbf{0}\}$
```



(a) (b)

Fig. 2: ROC curves: (a) Hand and face detection: 3-classes *TCAS* (left hand, right hand and faces) vs a 1-class cascade (faces) for frontal face detection, and (b) Humanoid and Aibo Detection.

at the first layer of the *TCAS* grows logarithmically with the number of classes, showing that feature evaluations are efficiently shared, reducing the computational complexity.

Experiment 3: robot detection. Finally, the proposed classifier is evaluated in a multiview and in a multiclass robot detection problem. SONY AIBO ERS7 robots are detected under 3 different views: frontal, lateral and back. The multi-class detection problem includes the multiview detection of AIBO robots plus a single-view of HAJIME HR18 humanoid robots. Altogether, four classes are defined (3 associated with AIBOs and one with HAJIME robots) and we built 4 one-class cascade detectors for each of the 4 classes (frontal, lateral, and back AIBOs, and Humanoids), a 3-classes *TCAS* is used to detect the AIBOs under the 3 views, and a 4-classes TCAS classifier

Table 2: *TCAS* and number of classes. Multiview RIP detection. Each view covers a 18° range. Avg. running time for images of 320x240 pixels. * value summed up over the 5 cascades.

Classifier	Training [min]	Running [sec]	Weak classifiers: Total	Weak classifiers: First layer
1x 20-classes	1862	8.05	10037	23
2x 10-classes	461	5.76	6712	20
4x 5-classes	176	6.10	3913	17
4x5x 1-class	75*	9.64	3251*	78*

that solves the defined multi-class robot detection problem. The classifiers are evaluated on 2 databases: AIBO DB and Humanoid DB (see Tab. 3 for details).

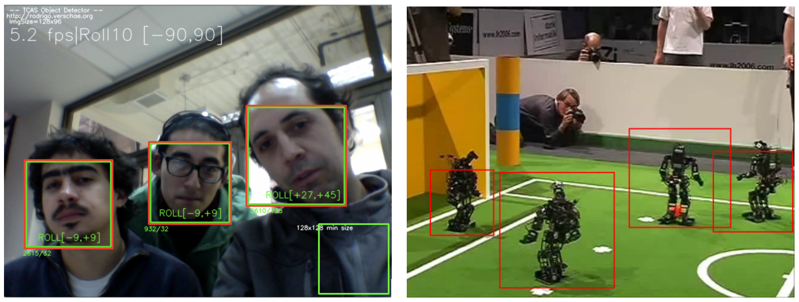
Table 3 (first two rows) presents the processing times for the multi-view AIBO detection problem. The obtained *TCAS* is 1.6 times faster than using the 3 one-class cascades in parallel. Fig. 3 (b) shows a detection example of using the 4-classes *TCAS* in the detection of HAJIME humanoids. Fig. 2 (b) presents ROC curves when the 4-class *TCAS* and a one-class cascade classifier are used for the detection of HAJIME robots only. Table 3 (last four rows) presents the processing time of the 4-classes *TCAS* used to detect all four classes, and the use of four one-class cascades. The 4-classes *TCAS* used to detect HAJIME humanoids has a clearly better performance than the cascade trained to detect just that class, with a gain of up to 17 percentual points for 0 false positives. In addition, the processing speed of both classifiers is very similar, with the *TCAS* being slightly slower when compared to the one-class cascade for humanoid detection. More importantly, when the 4-classes *TCAS* is used to detect Humanoids, and AIBOs (with frontal, lateral and back views), its processing speed is two times faster than the one required by four one-class cascades performing the same task.

Table 3: Robot detection: 3 views of Aibos robots (frontal, lateral and back) and Humanoids. Processing times on the HumanoidDB (640x480, 244 images) and AiboDB (208x160, 724 images)

Classifier	Target views	Database	Processing Time [sec]
3-class TCAS	Frontal, Lateral and Back AIBOs	Aibo DB	0.079
3 Parallel Cascades	Frontal, Lateral and Back AIBOs	Aibo DB	0.126
4-class TCAS	Humanoids	Humanoid DB	0.464
1-class Cascade	Humanoids	Humanoid DB	0.411
4-class TCAS	AIBOs & Humanoids	Humanoid DB	0.870
4 Parallel Cascades	AIBOs & Humanoids	Humanoid DB	1.767

5 Conclusions

In the present paper, a methodology for building multiclass object detectors was proposed. The proposed *TCAS* classifier corresponds to a multiclass classifier with a nested tree structure that makes use of coarse-to-fine (CTF) multiclass nested cascades and variants of multiclass weak classifiers in combination with a CTF tree structure.



(a) Face detection with view estimation (b) Humanoid Detection

Fig. 3: Robot and face detection examples. View estimation for in-plane rotations in $[-90, 90]$

TCAS was used to build multiclass and multiview objects detectors for three different problems. The obtained *TCAS* are efficient, being 1.5 to 2 times faster than the use of parallel cascades in all problems, and having the same or a lower number of false positives (for the same detection rate). Regarding the training time, the *TCAS* can be built efficiently, with training times of only hours for problems with a few classes (3-5) and mid-size training sets (5,000 samples per class) on modern workstation computers.

Future research directions include: (1) automatically grouping the classes instead of using predefined groups, (2) evaluating the system using other features, and on problems (e.g car and pedestrian), and (3) testing using larger number of classes.

References

1. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition. (2001) 511–518 1, 2
2. Wu, B., AI, H., Huang, C., Lao, S.: Fast rotation invariant multi-view face detection based on real adaboost. In: 6th Int. Conf. on Face and Gesture Recognition. (2004) 79–84 1, 2
3. Torralba, A., Murphy, K.P., Freeman, W.T.: Sharing visual features for multiclass and multiview object detection. IEEE Transactions on PAMI **29** (2007) 854–869
4. Huang, C., Ai, H., Li, Y., Lao, S.: High-performance rotation invariant multiview face detection. IEEE Trans on Pattern Analysis and Machine Intell. **29** (2007) 671–686 2
5. Wu, B., Nevatia, R.: Cluster boosted tree classifier for multi-view, multi-pose object detection. In: Proc. 11th IEEE Int. Conf. on Computer Vision (ICCV'07). (2007) 2
6. Freund, Y., Mason, L.: The alternating decision tree learning algorithm. In: Proc. 16th International Conf. on Machine Learning. (1999) 124–133 2
7. Verschae, R., Ruiz-del-Solar, J.: Multiclass adaboost and coupled classifiers for object detection. In: CIARP. Volume 5197 of LNCS., Springer (2008) 560–567 2, 4
8. Schapire, R., Singer, Y.: Improved boosting using confidence-rated predictions. Machine Learning **37** (1999) 297–336 4
9. Verschae, R., Ruiz-del-Solar, J.: Coarse-to-fine multiclass nested cascades for object detection. In: Int. Conference on Pattern Recognition. (2010) 344–347 4, 5, 7
10. Rowley, H.A., Baluja, S., Kanade, T.: Neural network-based detection. IEEE Transactions on Pattern Analysis and Machine Intelligence **20** (1998) 23–28 8