Tesis para optar al grado de Doctor en Ingeniería Eléctrica

Object Detection Using Nested Cascades of Boosted Classifiers

A learning framework and its extension to the multi-class case

March 25, 2010

Rodrigo Verschae

Profesor Guía: Javier Ruiz-del-Solar

Departamento de Ingeniería Eléctrica, Universidad de Chile, Chile

Abstract

Building robust and fast object detection systems is an important goal of computer vision. An approach that has proven to be effective is the use of cascade classifiers, approach introduced by Viola and Jones that allows to obtain fast detection results. Taking this work as starting point, methods for building one-class and multi-class object detectors are proposed. The contributions of this thesis work are three fold. First, an efficient learning framework of nested cascades of boosted classifiers for one-class object detection problems, with special emphasis on the training procedures, is proposed. Secondly, the proposed methods are extended to the multi-class case. Finally a nested tree of cascades, so called *TCAS*, is introduced, classifier that allows to better deal with multiclass object detection problems.

Most existing learning frameworks have a high computational complexity and require to predefine the structure of the classifier. The proposed learning framework for the one-class object detection problem is designed to build robust and fast object detectors in an efficient way without the need to predefine the structure of the classifier. This framework allows to build detection systems that have high accuracy, robustness, processing speed, and training speed. Using this framework state of the art face detection, eyes detection, and gender classification systems, as well as car, hand and robot detections systems are built.

When building systems that require the detection of multiple objects, the processing time becomes an important issue, as it increases linearly if several classifiers are used in parallel. Seeking to find a solution to this problem, we extend the concept of nested boosted cascade classifiers to the multiclass case. For this, a method to build multiclass nested cascades of boosted classifier is proposed, the use of coupled components in multiclass weak classifiers is introduced and compared to two existing alternatives, and the use of coarse-to-fine multiclass cascades in the object target space is proposed.

To better deal with the multiclass detection problem, a new classifier structure that consists of a nested tree of coarse-to-fine multiclass boosted cascades, so called *TCAS*, is proposed. The *TCAS* classifier is evaluated by building a multiclass detector of hand fists and faces, and a in-plane rotation invariant multiview face detector for which state of the art results on the CMU rotated database are obtained. Results showing that the proposed system is faster and more accurate than parallel cascades, and that it scales well with the number of classes, during training and running time, are presented. The results show the robustness of the proposed methods for building multiclass and multiview object detectors.

Acknowledgements

First, I want to express my gratitude to Javier Ruiz-del-Solar, my thesis advisor. He has been a great guide, collaborator and friend. This thesis would not have been completed without his continuous encouragement, support and guide. He has not only supported me on choosing this topic, but he also gave me the freedom to explore and test many –good and bad– ideas. His sound advice, guidance, insights and intuitive view helped to narrow and focus this search. He also imbibed in me the importance of publishing, careful experimentation, and pushing the development of the algorithms to the stage of being usable in real world conditions. Finally, his disposition and will to guide me remotely during my stays abroad is something I really appreciate.

Thanks to the thesis proposal review committee, Pablo Estevez, Richard Weber and Doris Sáez. Their feedback and detailed comments helped to narrow and focus the contents of the thesis topic.

Special thanks to Doris Sáez, Alvaro Soto, and Jean-Philippe Thiran for taking their time to be part of the thesis review committe. Their insight and remarks helped to improve the present document.

I would like to thank Andrés Almansa and Agnès Desolneux for guiding my work on a-contrario validation methods during my stay at CMLA, ENS de Cachan, France. I really appreciate their time, guide and support. The topic I developed with them was finally not included in the present document, mainly to keep it focused on the learning algorithms and not to make it even longer; this was not an easy decision, but I think it was the right one.

I would also like to thank Gregory Randall and all the people organizing and coordinating the ALFA network project in computer vision, which not only allowed me to do two research visits at ENS Cachan, and to attend the workshops that were organized, but also to meet many people working on related topics. I think that the friendships and networks created are already a sign of the success of this project and that they will become even more fruitful in the future.

I am particularly grateful to Rafael Grompone von Gioi, José Verschae –my bother–, and Mauricio Correa, not only for their friendship, but also for their support and help during different stages of my work. Rafa's comments on this work, and our discussions help me to clarify some concepts and also broaden my view in related topics. José was of great help, in particular with some optimization issues, as well with his help proof-reading a part of this document. Mauricio, besides collaborating in some experiments, and his feedback on the performance of the algorithms, was of great help with some paperwork and on maintaining the servers while I was not in Chile.

I would also like to thank Mario Köppen for his friendship, insights, comments and feedback. Our joint project at the beginning of this work pushed it to be of practical use in real world applications. I also really appreciate his invitation to work at Kyushu Institute of Technology, Japan.

I would also like to thank Aureli Soria-Frisch for receiving me during my research visit at the Pompeu Fabra University, and for his useful comments, kindness and friendship.

I can not forget to mention Anna Ukovich, Raul Vicente-Garcia, Gabriele Facciolo, Enric Meinhart-Llopis, Filippo Santambrogio, Jeremi Jakubowicz, Nguyen Hai Yen, Camilo Melani,

Fabian Munoz, José delpiano, Patricio Loncomilla, Pablo Zorzi, Kaori Yoshida, and many others. Their friendship and encouragement during my work, stays, and short visits at the Fraunhofer IPK (Germany), ENS de Cachan (France), Universidad de Chile (Chile), Pompeu Fabra University (Barcelona) and Kyushu Institute of Technology (Japan) made this work much more pleasant.

Thanks to Hardy Franke, Oscar Sanhueza, Matias Arenas, and Jong Bor-Lee for their feedback after using some of the algorithms presented here.

I would also like to thank the secretaries Eliana Monardes at Universidad de Chile, and Veronique Almadovar at ENS de Cachan. I really appreciate their help and good humor.

I can not end without thanking my family, in particular my parents for their absolute confidence on me, and for their continuous support and encouragement. I would also like to thank them for driving and picking me up from the airport each and every time, regardless of the flight time.

Finally I would like to thank and acknowledge all the programs and institutions that partially supported this work:

- CONICYT Chile, doctoral fellowship program (2005-2008).
- The European ALFA Network in Computer Vision Foundations and Applications, project number II-0366-FA (2005-2006, 2007).
- Vice-rectoría de Asuntos Académicos (VAA), Universidad de Chile, Chile, short research visits program (2008).
- The MECESUP2 program, Ministerio de Educación, Chile (2009).

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Contributions

In the present chapter the original contributions of the present work on object detection are briefly outlined and the relevant publications of this and other works done at the Universidad de Chile while being a doctorate student are presented.

Development of a learning framework for one-class object detection

- The integration of powerful learning capabilities together with effective and fast training procedures, which allows building nested cascades for object detection problems that have high accuracy, robustness, and processing speed.
- These learning methods and training procedures include: the use of internal and external bootstrapping, the use of domain partitioning weak classifiers, and the use of effective heuristics to speed up the training process. As a result, the training time only takes hours on a standard computer, which is an important gain compared to days or weeks required by related works.
- The definition of a criteria for handling the trade-off between the processing speed, the true positive rate and the false positive rate during the training. This is done by automatically selecting the number of weak classifiers at each layer of the cascade. Therefore, and unlike most previous works, it is not needed to predefine the structure of the cascade.
- Using this system hand detection, robot (humanoid and Aibo) detection, profile car detection, face detection, eyes detection, and gender classification systems are built. The performance of these systems is validated and analyzed using standard face databases (BioID, FERET and CMU-MIT), and car database (UIUC), where state of the art results are obtained.

Extension of the learning framework to the multi-class case

- The extension of nested cascades to multiclass nested cascades and the required training procedures. This includes the introduction of a new way of training multiclass weak classifiers using coupled components and the identification of two existing ones (that we call joint components and independent components).
- Use of coarse-to-fine search on the object target space in multiclass cascades, which allows to build faster and more accurate multiclass nested cascades.

- The extension of the multiclass nested cascades to a tree structure (so called *TCAS*), which consist of a coarse-to-fine nested tree of multiclass nested cascades.
- The proposed training procedures are scalable and efficient, requiring a reasonable training time (considering the complexity of the problem).
- Best results are obtained when weak classifiers with coupled components are used at first levels of the *TCAS* and independent components in the later layers.
- The processing time of the obtained multiclass *TCAS*'s for *n* classes (or views) is shorter than running *n* 1-class cascades in parallel, and their accuracy is slightly higher.
- Using the proposed framework, a state of art the multiview face detector invariant to rotations-in-plane (RIP) was built, as well as an accurate and fast multiclass hand and face detector.

Publications

The publications prepared while pursuing the doctorate at the computer vision laboratory of the Universidad de Chile that are directly related to this thesis work are outlined. Most of the methods and results presented on these publications are included, at least partially, in the present document. These include one book chapter, one journal (ISI) article, and two LNCS papers. The outline of these publications is:

- [Verschae and Ruiz-del-Solar, 2006] : This book chapter summarizes a tutorial on boosted cascade methods presented at the 2005' *World Conference on SoftComputing (WCSC)*. This publication is the base of Chapter 2 and 3.
- [Verschae et al., 2008b][a] (ISI) presents a framework that was used for building one-class object detectors. The main contributions have to do with the proposed training procedures, which allow to have a shorter training time (about a day on a single computer) compared to similar systems (where the training time takes months or weeks on a single computer) and where the complexity of a layer of the cascade (number of weak classifiers) does not need to be preset. Using this framework a face detector, an eye detector and a gender classifier are built. The performance of the face detector is among the best published results in terms of accuracy and processing time. In the case of the eye detector, to the best of our knowledge, the obtained results are the best published so far. This publication is the base of Chapter 4.
- [Verschae et al., 2006] (ISI): This article presents a gender classification system. It was built using the framework used to develop the face detector described in [Verschae et al., 2008b]. This gender classifier shows very robust results when used in a real world application, i.e., when used together with a face detector and an eye detector.
- [Verschae and Ruiz-del-Solar, 2008] proposes an extension of the boosted cascade classifiers presented in [Verschae et al., 2008b] to the multiclass case, with the goal of building multiclass and multiview object detectors. We propose to use nested cascades of multiclass boosted classifiers, and introduce the concept of coupled components in multiclass classifiers. This publication presents a preliminary solution for the multiclass detection problem using cascades classifiers and is the base of the first part of Chapter 5 (Sections 5.1 to 5.3).

[a]The Machine Vision and Applications (MVA) journal had an impact factor of 1.485 in 2008.

Note that some results on hand and robot detection published in [Franke et al., 2007] and [Arenas et al., 2007] are presented in section 4.4.2.Those papers describe large systems that make use of some methods proposed here.

Other Publications

I contributed to different projects at the Universidad de Chile during the Doctorate work and the research visit at ENS de Cachan, and not all contributions were included on the present document. Publications related to projects which are not part of this thesis work include: one journal (ISI) article, 3 LNCS papers, and 6 conference papers. The outline of these publications is:

- [Verschae et al., 2005]: Improvement of a face detector using a multi-objective genetic algorithm.
- [Ruiz-del-Solar et al., 2007]: Face analysis for Human-Computer-Interaction (HCI) applications.
- [Franke et al., 2007]: Hand detection and gesture recognition.
- [Arenas et al., 2007]: Robot (Humanoid and Aibo) detection.
- [Ruiz-del-Solar et al., 2009b] (ISI): Journal article presenting a study about face recognition in unconstrained environments for Human-Computer-Interaction (HCI). A preliminary version of this work was presented in [Verschae et al., 2008a].
- [Hermosilla et al., 2009]: Face recognition using thermal (infrared) images.
- [Ruiz-del-Solar et al., 2009a]: Analysis of a social robot in different naturalistic environments.
- [Lee-Ferng et al., 2010, Ruiz-del-Solar et al., 2009d, Ruiz-del-Solar et al., 2009c]: Dynamic hand gesture recognition, and hand gestures recognition for Human-Robot-Interaction (HRI).

Articles under review or preparation

- A journal article presenting the methods and results of Chapter 5 on multiclass object detection is under preparation.
- In the technical report prepared at ENS Cachan [Verschae, 2006] we present preliminary work on how a careful modeling of the search procedure, performed in the *overlapping detection processing module* (the last module of the figure 2.1) allows to obtain better detection results. This was done by introducing a-contrario models of the classifier and of the search procedure. This allowed to robustly determine the final detections and to control the number of false detections. An article presenting several improvements of this work is under preparation.
- Journal article [Arenas et al., 2009] presenting an extended version of [Arenas et al., 2007] presenting not only robot (Humanoid and Aibo) detection, but also robot refereeing and the use of context in the detection (submitted, currently being revised).

Oral presentations in international conferences and workshops

- Face Detection using boosted nested cascades. School on Neuromathematics of Vision, De Giorgi Research Center, Pisa, Italia, 4 - 9 Septiembre, 2006.
- Face Recognition in Unconstrained Environments: A Comparative Study. Faces in Real-Life Images: Detection, Alignment, and Recognition workshop at ECCV 2008, Marseille, France, October 17, 2008.
- Multiclass object detection: nested cascades, vectorized Adaboost and coupled components. ALFA workshop in Computer Vision Foundations and Applications, ENS de Cachan, France, 10-11 March, 2008.

List publications done during the doctorate studies

Publications related to the thesis

- [Verschae and Ruiz-del-Solar, 2006]: Verschae, R. and Ruiz-del-Solar, J. (2006). Applications of Soft Computing: Recent Trends, volume 36, chapter State of the Art Face Detection: Cascade Boosting Approaches. Springer.
- [Verschae et al., 2005]: Verschae, R., Ruiz-del-Solar, J., Köppen, M., and Vicente-Garcia, R. (2005). Improvement of a face detection system by evolutionary multi-objective optimization. In Proc. 5th Int. Conf. on Hybrid Intelligent Systems, pages 361366.
- [Verschae et al., 2006]: Verschae, R., Ruiz-del-Solar, J., and Correa, M. (2006). Gender classifica- tion of faces using adaboost. In Lecture Notes in Computer Science 4225 (CIARP 2006), pages 6877.
- [Verschae et al., 2008b]: Verschae, R., Ruiz-del-Solar, J., and Correa, M. (2008b). A unified learn- ing framework for object detection and classification using nested cascades of boosted classi- fiers. Machine Vision Applications, 19(2):85103.
- [Verschae and Ruiz-del-Solar, 2008]: Verschae, R. and Ruiz-del-Solar, J. (2008). Multiclass ad- aboost and coupled classifiers for object detection. In Ruiz-Shulcloper, J. and Kropatsch, W. G., editors, CIARP, volume 5197 of Lecture Notes in Computer Science, pages 560 - 567. Springer
- [Franke et al., 2007]: Franke, H., del Solar, J. R., and Verschae, R. (2007). Real-time hand ges- ture detection and recognition using boosted classifiers and active learning. In Mery, D. and Rueda, L., editors, PSIVT, volume 4872 of Lecture Notes in Computer Science, pages 533547. Springer.
- [Arenas et al., 2007]: Arenas, M., del Solar, J. R., and Verschae, R. (2007). Detection of aibo and humanoid robots using cascades of boosted classifiers. In Visser, U., Ribeiro, F., Ohashi, T., and Dellaert, F., editors, RoboCup, volume 5001 of Lecture Notes in Computer Science, pages 449456. Springer.
- [Ruiz-del-Solar et al., 2007]: Ruiz-del-Solar, J., Verschae, R., Vallejos, P., and Correa, M. (2007). Face analysis for human computer interaction applications. In Ranchordas, A.,

Araujo, H., and Vitria, J., editors, VISAPP (Special Sessions), pages 2330. INSTICC - Institute for Systems and Technologies of Information, Control and Communication.

- [Ruiz-del-Solar et al., 2009c]: Ruiz-del-Solar, J., Verschae, R., Correa, M., Lee-Ferng, J., and Castillo, N. (2009c). Real-time hand gesture recognition for human robot interaction. lecture notes in computer science (robocup symposium 2009) (in press).

Other peer-reviewed publications done at the Universidad de Chile

- [Verschae et al., 2008a]: Verschae, R., Ruiz-Del-Solar, J., and Correa, M. (2008a). Face Recogni- tion in Unconstrained Environments: A Comparative Study. In Workshop on Faces in Real- Life Images: Detection, Alignment, and Recognition, Marseille France. Erik Learned-Miller and Andras Ferencz and Frederic Jurie.
- [Ruiz-del-Solar et al., 2009b]: Ruiz-del-Solar, J., Verschae, R., and Correa, M. (2009b). Recog- nition of faces in unconstrained environments: A comparative study. EURASIP Journal on Advances in Signal Processing, 2009.
- [Hermosilla et al., 2009]: Hermosilla, G., Ruiz-del-Solar, J., Verschae, R., and Correa, M. (2009). Face recognition using thermal infrared images for human-robot interaction applications: A comparative study. In LARS 2009, Oct. 29 - 30, Valparaso, Chile (CD Proceedings).
- [Ruiz-del-Solar et al., 2009a]: Ruiz-del-Solar, J., Mascaro, M., Correa, M., Bernuy, F., Riquelme, R., and Verschae, R. (2009a). Analyzing the human-robot interaction abilities of a general- purpose social robot in different naturalistic environments. In Lecture Notes in Computer Sci- ence (RoboCup Symposium 2009) (in press).
- [Ruiz-del-Solar et al., 2009d]: Ruiz-del-Solar, J., Verschae, R., Lee-Ferng, J., and Correa, M. (2009d). Dynamic gesture recognition for human robot interaction. In LARS 2009, Oct. 29 - 30, Valparaso, Chile (CD Proceedings).
- [Lee-Ferng et al., 2010]: Lee-Ferng, J., Ruiz-del-Solar, J., Correa, M., and Verschae, R. (2010). Hand gesture recognition for human robot interaction in uncontrolled environments. In Work- shop on Multimodal Human - Robot Interfaces, 2010 IEEE International Conference on Robotics and Automation May 3, 2010, Anchorage, Alaska, (Accepted).

Chapter 2

Introduction

“Éste, no lo olvidemos, era casi incapaz de ideas generales, platónicas. No sólo le costaba comprender que el símbolo genérico perro abarcara tantos individuos dispares de diversos tamaños y diversa forma; le molestaba que el perro a las tres y catorce (visto de perfil) tuviera el mismo nombre que el perro de las tres y cuarto (visto de frente).”

Excerpt from *“Funes el memorioso”* by *Jorge Luis Borges*.

Contents

2.1 Motivation

An important goal of the computer vision community is to develop systems that can detect objects appearing in cluttered backgrounds of arbitrary natural images. In the following *object detection* will be understood as *finding*[a] all objects of a given (set of) *class(es)*[b], if any, appearing in an image.

Object detection problems can be classified in two cases: the detection of *low level* objects, such as geometrical figures, where the objects correspond to lines, segments, rectangles, circles, etc., and the detection of *high level* objects, where the objects correspond to faces, cars, hands, etc. We will focus on the second case.

Object detection is a complex problem, mainly because of *i*) the variability within the objects of the same class, *ii*) the variability in the process of the image acquisition, (variable illumination, rotations, translations, scales, distortions, etc.), *iii*) the variability in the images's background, and *iv*) the processing time constrains of many applications.

The object detection problem has been widely studied and good results have been obtained for some particular cases. Among many object detection problems, face detection has concentrated most of the attention, because in many different situations it an important cue to detect persons in a given scene, and it is a key step in almost any computational task related with the analysis of faces in digital images.

[a]Herein *finding* means to determine the location (and size) of the objects appearing in the image.
[b]Herein *class* is defined as a particular *type* (or *kind*) of object.

2.2 Challenges in Object Detection

Although physical objects usually have a 3D structure, most of the existing methods for object detection only deal with the detection of objects under a single view or pose.[c] For detecting objects under different poses or views, most methods perform different and separated searches, each time considering a given view/pose of the object. When using this approach, if more than one class of object is to be detected -or the same class of object under different views/poses is to be detected- , a detector for each class of object is used, and consequently the processing time is the sum of the time required to process the image with each detector. Therefore this kind of system is not scalable with the number class of objects to be detected.

During the last years a few works (such as [Wu et al., 2004] [Schneiderman, 2004], [Jones and Viola, 2003b] and in particular [Torralba et al., 2007] [Huang et al., 2007] [Wu and Nevatia, 2007]), methods based on cascade classifiers, have tried to overcome this problem obtaining promising results (see Chapter.3). Although these methods go in the right direction, we think that they can be improved. These methods present some problems that still have to be solved: some of them ([Jones and Viola, 2003b] [Schneiderman, 2004] [Wu et al., 2004]) do not exploit the similarities between the different objects or views, others ([Torralba et al., 2007]) do not make use of coarse-to-fine classifiers, and some others ([Torralba et al., 2007] [Wu et al., 2004] [Jones and Viola, 2003b]) do not exploit training procedures (like internal bootstrapping), or do not do it in the best way ([Huang et al., 2005] [Li and Zhang, 2004] [Kölsch and Turk, 2004] [Wu et al., 2004]). Also some systems are specifically designed for multiview and multipose problems ([Huang et al., 2007] [Wu and Nevatia, 2007]), and not for multiclass problems.

In the present thesis work we are interested in developing a framework for building systems that can detect different kinds of objects at the same time. It is important to add that the system is to be designed to work on grey scale images, not on color ones. The work will first focus developing robust one-class object detectors and the developing and integration of the required learning methods and training procedures. Later we focus on the multiclass object detection problem. Most analysis will be done using the problem of frontal face detection in the one-class case, and on the detection of faces seen from different views, which can be handled as a multiclass problem. We use faces as base problem because most existing methods have been tested on this problem. Nevertheless we also evaluate the proposed methodology on other classes of *high level* objects, such as eyes, hands or cars. The evaluation will be done in terms of error rates, processing speed, scalability of the detection system, training speed, and scalability of the training procedure.

2.3 State of the Art

Several approaches have been proposed for the detection of objects in digital images. Most of these approaches have been studied on the face detection case. A comprehensive review (of methods up to the year 2002) can be found in [Hjelms and Low, 2001] and [Yang et al., 2002]. Following these reviews, existing approaches can be classified under the following categories:

- knowledge-based, which are based on rules encoded by humans,

[c]Herein *view* refers to a particular range of view-angles of the object with respect to the capturing device. Note that *pose*, although sometimes used as synonym of *view*, will be used to refer to different configurations of the object, e.g. a hand can have many different poses – fist, peace sign, etc – it can also have many different views – frontal, lateral, etc.

- template-based, which are based on template matching or shape models,
- feature-based, which are based on the analysis of low-level features (e.g. color, edges, textures), and
- appearance-based (also called image-based), which employ statistical methods like projections in subspaces, statistical analysis or statistical classifiers (e.g. neural networks).

Appearance-based approaches, in particular the ones based on statistical classifiers, have shown a higher performance than the knowledge-based, template-based and feature-based ones ([Hjelms and Low, 2001] [Yang et al., 2002]). When using appearance-based approaches, the common procedure [Hjelms and Low, 2001] [Yang et al., 2002] for detecting all instances of a type of object is to perform an exhaustive search over different scales and positions of the image. For this, a statistical classifier is trained to discriminate between patches of the image containing the background and the object to be detected. These patches are commonly called *windows* and they are of fixed size (e.g. 24x24 pixels). Therefore, the classifier is trained to discriminate between object and non-object windows for a fixed window size. This approach is sometimes referred to as the *sliding windows* approach.

In the present section a description of the commonly used architecture of detections systems based on statistical classifiers is given (see section 2.3.1), followed by a description of existing methods used on the different modules of this kind of system (see sections 2.3.2, 2.3.3, 2.3.4 and 2.3.5).

2.3.1 Detection Architecture

Most object detection systems based on statistical classifiers require to perform an exhaustive search over the different positions and scales of the image in order to detect objects that appear at different positions and scales. This approach is commonly called the sliding-window approach. The detection architecture of this kind of system is shown in Figure 2.1 and it works as follows:

1. In order to detect faces at different scales, the input image is downsized several times obtaining a pyramid of images that represent the input image at multiple resolutions (*Multiresolution Analysis* module).
2. In the *Window Extraction* module, windows of a fixed size (e.g. 24x24 pixels) are extracted from each scaled version of the input image.
3. In the *Pre-processing* module, each extracted window can be processed (e.g. variance normalization or histogram equalization) for obtaining some invariance against illumination changes prior to the classification.
4. Each window is analyzed by a classifier (*Classification Module*) that predicts if it is an object window or a non-object window.
5. After all considered windows have been processed and classified, in the *Overlapping Detection Processing module* the size and position of the final detections are determined.

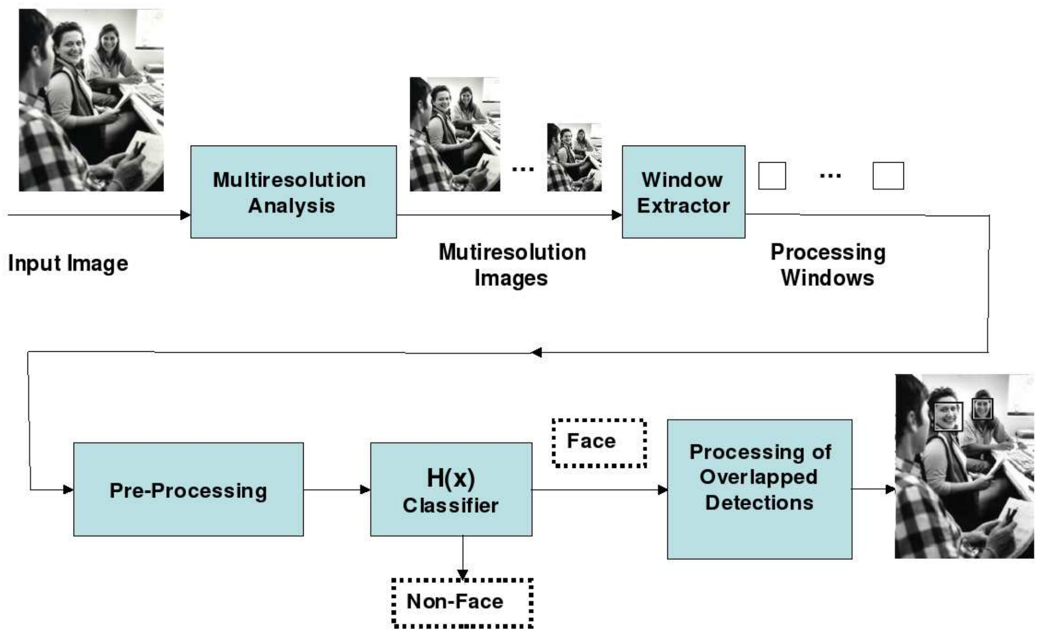


Figure 2.1: Block diagram of an object detector (exemplified for the case of a face detector)

2.3.2 Representation

In any pattern recognition problem an important issue is data representation. For example, the representation can be based on features (measures on the input data), or it can be *holistic* (taking the input data as a whole). In the case of the object detection problem solved using statistical classifiers, different kinds of representations have been used:

1. Early representations used in face detection were based on the pixel intensities or on simple operations on groups of pixels. In [Rowley et al., 1998] a representation that tries to resemble the retinal connectivity of the human visual system is used; the inputs to an artificial neural network correspond the average of different groups of adjacent pixels. In [Yang et al., 2000b] a binary sparse representation which jointly codes the pixels intensity and position is considered.

2. In [Osuna et al., 1997], [Romdhani et al., 2001] and [Yang et al., 2000a] a holistic representation of the input image is used. [Osuna et al., 1997] and [Romdhani et al., 2001] use the holistic representation together with a SVM classifier, while [Yang et al., 2000a] uses a method based on projection of the input window on a given subspace.

3. In [Papageorgiou and Poggio, 2000] the used input feature vector is inspired on the Haar wavelet transform. The system use a SVM classifier and the authors were able to build pedestrian, car and face detectors. In [Viola and Jones, 2004] (first published in [Viola and Jones, 2001]by the same authors), Haar-like wavelet features that can be evaluated very fast using the integral image were introduced. This representation, also called rectangular features, has been extensively used on the object detection problem [Viola and Jones, 2002] [Viola et al., 2003] [Lienhart et al., 2003] [Xiao et al., 2003] [Wu et al., 2004] [Lin et al., 2004] [Verschae et al., 2005] [Fasel et al., 2005], among many others. A similar representation that was used by [Li and Zhang, 2004] and [Kölsch and Turk, 2004], corresponds to a over-complete base

of features which can also be evaluated using the integral image. Most of these features do not longer resemble Haar wavelet transform and they are so many that training time is much more longer than [Viola and Jones, 2004](several months on a single computer).

4. In [Fröba and Ernst, 2004] is introduced the mLBP (modified Local Binary Patterns)[d] features in the object detection problem. The mLBP representation is invariant to linear contrast changes, and can be evaluated with a few simple operations.

5. In [Schneiderman and Kanade, 2000], [Schneiderman, 2004] and [Liu and Shum, 2003] a representation based on the coefficients of the wavelet transform is used. Although this kind of representation is powerful, it has the disadvantage of being computationally expensive.

6. In a recently published work [Villamizar et al., 2006] use approximated steerable filters [Freeman and Adelson, 1991] for calculating the features is proposed. Steerable filters are rotated filters that can be expressed as a linear combination of base filters. In [Villamizar et al., 2006] is proposed to approximate these filters using rectangular features that can be evaluated efficiently using the integral image [Viola and Jones, 2004].

7. In [Vidal-Naquet and Ullman, 2003] and [Torralba et al., 2007] features based on part of objects are used. Each feature is obtained by evaluating a measure that is similar to the correlation of the window being analyzed with a patch of a windows containing the objects. These patches are selected during training using a boosting algorithm.

2.3.3 Pre-processing

Systems like the ones presented in [Rowley et al., 1998] [Osuna et al., 1997] [Romdhani et al., 2001] and [Yang et al., 2000a] require each input window to be pre-processed in order to obtain good results. These pre-processing procedures are similar to the ones used on face recognition systems [Ruiz-del-Solar and Navarrete, 2005] and add a considerable computational time. They are used to normalize the data for obtaining some invariance against illumination changes before performing the classification. Usually more than one of them is used. Existing pre-processing procedures include: (a) normalization, (b) standardization, (c) histogram equalization, (d) best plane subtraction, (e) masking, (f) self-quotient image estimation [Wang et al., 2004], and (g) the use of total variation models [Chen et al., 2006].

Most recently proposed systems (such as [Viola and Jones, 2004] [Wu et al., 2004] [Schneiderman, 2004]) , including the ones used in the present thesis work, do not requite the input window to be preprocessed, because the used features are illumination invariant to a large degree. Therefore these last methods require less processing time, allowing a faster detection.

2.3.4 Statistical Classifiers

As previously mentioned, many different appearance-based have been used in the problem of object detection [Hjelms and Low, 2001] [Yang et al., 2002]. Starting with the seminal works [Sung and Poggio, 1998] [Rowley et al., 1998], both based on neural networks, and [Osuna et al., 1997], which uses a SVM, successful appearance-based approaches include the use of SNoW classifiers [Yang et al., 2000b], Bayesian classifiers [Schneiderman and Kanade, 2000], neural networks [Delakis and Garcia, 2004], and coarse-to-fine classifiers [Viola and Jones, 2004] [Viola

[d]LBP (Local Binary Patterns) are also known as texture numbers and census transform.

Reference	DR	FP	Processing Time & image size	Methodology
[Rowley et al., 1998]	83.2	10	1000ms, 320x200	Neural Network
[Delakis and Garcia, 2004]	90.5	10	700ms, 330x240	Convolutional Neural Network
[Schneiderman, 2004]	89.7	6	180ms, 320x200	Feature-Centric Cascade
[Wu et al., 2004]	90.1	10	80ms, 320x240	Boosted Nested Cascade

Table 2.1: Face detection results on the MIT-CMU Face Database (130 images, 507 frontal faces) for some selected methods. The Detection Rate (DR), number of false positives (FP), average processing time reported by the authors and methodology are presented.

and Jones, 2002] [Liu and Shum, 2003] [Wu et al., 2004] [Fleuret and Geman, 2001] [Schneiderman, 2004] [Sahbi and Geman, 2006] [Romdhani et al., 2001] (see Chapter 3 for more details on coarse-to-fine classifiers). Example results taken from the literature for four face detector systems are shown in Table 2.1. This kind of methods (appearance-based [Hjelms and Low, 2001] [Yang et al., 2002] methods based on statistical classifiers) have been the most succesfull ones.

Object Recognition

Object recognition, sometimes called object categorization, is commonly understood as determining whether the image under analysis contains an object belonging to a particular category. Many approaches have been developed for the problem of object recognition and some of them can be used on the problem of object detection. The main goal of object recognition is determining the presence of the object in the image, and not to determine the location and size of all object instances as required in the object detection problem. Determining the presence of an object can be achieved, for example, by methods that can localize only one object of the category, or by using representations that encode information of the image as a whole (e.g. not considering the spatial information of the features).

The literature on object recognition is bast. Thus in the following we only outline the main existing research lines on this topic, and we give a few examples of methods that have been used on both object detection and object recognition. Object recognition methods can be roughly categorized as: (a) part based (e.g methods based on sparse representations [Agarwal et al., 2004] [Mutch and Lowe, 2008]), (b) Bag of Words (BoW) (e.g [Vidal-Naquet and Ullman, 2003] [Larlus et al., 2009]), (c) discriminative Models (e.g [Moosmann et al., 2008] [Lampert et al., 2009]), and (e) combined segmentation and recognition (e.g. [Leibe et al., 2004]). More complete lists of references on object recognition can be find in [Mutch and Lowe, 2008] [Moosmann et al., 2008] [Ponce et al., 2006] [Lampert et al., 2009] and [Larlus et al., 2009].

2.3.5 Determining the Final Detections

After classifying all windows of the image, the final detections have to be determined. This is done in *the overlapping detection processing module* (the last module of Figure 2.1).

Usually, positive outputs of the classifier will occur more than once for some windows which are close in position and scale. Most times these positive outputs correspond to correct classifications of the same object at different scales and translations. In other words, different windows “containing” the same object will be classified as corresponding to the object and these detections will overlap.

Using this information the final detections can be determined. Not all works describe how this is done. Considering those that have explained it, we have categorized the method used to determine the final detections in two categories:

- Activation map: A 2-dimensional map (or 3-dimensional when the scale pyramid is considered) is constructed using the *confidence*[e] given by the classifier for each classified window. Afterwards the detections are selected using this map. In [Schneiderman and Kanade, 2000] the local maximums in the map are located and then thresholded, while in [Rowley et al., 1998] the sum of the confidence for subregions of the activation map are obtained and then it is tested if the sum is larger than a given threshold. In [Agarwal et al., 2004] a *neighborhood suppression* procedure is performed, which consist in searching, iteratively, for high-activation peaks in the activation map.

- Windows' grouping: In [Viola and Jones, 2004] [Delakis and Garcia, 2004] groups of windows classified as positive are formed. The groups are formed using a clustering algorithm (with a bottom up procedure) that uses as features the position of the windows, and stopping rules based on the scale of the windows. The groups are validated and considered as face if the number of windows is larger than a given threshold [Viola and Jones, 2004], and if the sum of the confidences is larger than a given threshold [Delakis and Garcia, 2004].

2.4 Proposed Research Work

During the last years, coarse-to-fine (CTF) classifiers, in particular the ones based on the boosted cascade approach, have drawn most of the attention regarding the development of object detection systems, because they allow to obtain very fast detection systems that are capable of achieving high detection rates.

In a recent paper [Gangaputra and Geman, 2006]), Gangaputra and Geman define and clarify the concept of coarse-to-fine classifiers and at the same time try to unify previous work on cascades and tree-structured classification:

"*Coarse-to-fine classification is an efficient way of organizing object recognition in order to accommodate a large number of possible hypotheses*[f] *and to systematically exploit shared attributes and the hierarchical nature of the visual world. The basic structure is a nested representation of the space of hypotheses and a corresponding hierarchy of (binary) classifiers.*"

One of the main reasons for using coarse-to-fine classifiers is to achieve a fast classification of the window (image patches). This fast classification is achieved by using classifiers that make the decision about whether the windows correspond to the object or not, by performing a sequence of questions (verification stages, or hypothesis testing). Each of these questions reduces the space of answers for the possible non-objects and these questions usually increase in complexity, i.e., the firsts questions require less processing time than the later ones. In this way, windows that can be easily classified as non-object are discarded by the first stages (first questions), while object-like windows are analyzed by several stages. This allows to obtain fast classifiers because of two reasons: (1) there is an important difference in the a priori probability of occurrence of the classes, i.e. there are much more non-object windows than object windows in any given image, and (2)

[e]The *confidence* of the classifier corresponds to the real value output given by the classifier. The module of *confidence* value indicates "how sure" is the classifier about the classification, the larger this value, the less likely the classification is incorrect.

[f]With *hypotheses* Gangaputra and Geman refer to classes, subclasses or specific instances of objects.

most of the non-object windows are quite different from the object, therefore they can be easily discarded, i.e., classified as non-object. Hence, the average processing time of a window is almost completely defined by the processing time of the non-object windows.

In the context of object detection several well know classifiers have been extended or adapted to build coarse-to-fine classifiers. Among them we can find boosted classifiers [Viola and Jones, 2004] [Viola and Jones, 2002] [Viola et al., 2003] [Wu et al., 2004] [Lienhart et al., 2003] [Brubaker et al., 2008], Bayesian classifiers [Schneiderman, 2004], projection based classifiers [Keren et al., 2001], SVM classifiers [Kienzle et al., 2004] [Sahbi and Geman, 2006], SNoW classifiers [Fröba and Ernst, 2004], and coarse-to-fine decision trees [Fleuret and Geman, 2001].

It is worth mentioning that although most works using coarse-to-fine classifiers deal with the problem of face detection, this kind of classifiers has been used in other detection problems, such as cars [Schneiderman, 2004], hands [Kölsch and Turk, 2004], moving pedestrians [Jones and Viola, 2003a], traffic signs [Schneiderman, 2004], eyes [Jones and Viola, 2003a] [Verschae et al., 2008b] [Fasel et al., 2005] and numbers [Amit et al., 2004].

2.4.1 Objectives

Main Objective

The main objective of the proposed thesis work is *to develop a framework for building fast and robust multiclass object detectors based on coarse-to-fine boosted classifiers*.

Specific Objectives

Taking as a base the main objective, specific objectives are defined:

- To develop a efficient training framework to build cascade object detectors
  - To define a criterion for training each stage of the cascade/coarse-to-fine classifier. This criterion should allow handling the trade-off between the processing speed and the performance, seeking to minimize the false positive rate, to minimize the average processing time and to maximize the true positive rate of the coarse-to-fine classifier.
  - To propose a procedure to select the negative training examples of each layer of the cascade.
- The training time should take less than a week (in a x686 2 GHz computer) when the cascade classifier is build for detecting one class of objects (under one view)[g].
- To extend boosted cascade classifiers to **coarse-to-fine boosted multiclass classifiers**.
  - The training procedure should be scalable: the training time should grow linearly with the number of training examples, linearly with the number of features, and at most quadratically with the number of classes.
  - The classifier structure should be scalable: the processing speed of the obtained classifier should grow sub-linearly with the number of classes.

[g]Systems like [Viola and Jones, 2004] require months to be trained in a single computer.

Some Comments on the Objectives

- It is not expected that the detection systems will detect all classes of objects with the same accuracy. The detection of some classes of objects is more difficult than others. Some of them are just more "complex" problems, and the training databases may contain less training examples or less representative training examples.

- Most of the existing systems use large training databases (e.g 20,000 positive training examples) to obtain high accuracy [Schneiderman, 2004] [Wu et al., 2004] [Huang et al., 2007]. We plan to train the classifiers using small and mid-size datasets.

- To test and evaluate the proposed algorithm, the problem of face detection will be used. The face detection problem has been widely studied and standard evaluation databases exist, which allows to compare the developed methods with state of the art systems.

- The work will be focused on the learning process and not on the features being used (we will make use of rectangular Haar-like features and mLBP features). In many works it has been shown that different features types are better fitted for different object detection problems. For example, compared the use of Haar-like features, it has been shown that in terms of accuracy is better to use extended Haar-like features [Lienhart et al., 2003] and Gaussian-based filters [Meynet et al., 2007] for frontal face detection, granular features [Huang et al., 2007] for multiview face detection, edgelet features [Wu and Nevatia, 2005] for car and pedestrian detection, and part-base correlation [Torralba et al., 2007] for many-classes detection problems.

2.4.2 Hypotheses

In the present section we outline the main hypotheses, and we give specific hypotheses on the tools and directions that will be used to solve the outlined problems.

Main Hypotheses

- It is possible to a developed training procedure to train nested cascades classifiers that is efficient in terms of computational complexity and that allow to obtain robust detectors. This can be done without the need to preset parameters that predefine the structure of the cascade classifier.

- Existing coarse-to-fine methods can be extended to the multiclass case, or existing multiclass classifiers can be extended to coarse-to-fine classifiers, allowing to achieve a fast, robust, and accurate classification in the multiclass case.

Specific Hypotheses

- One-class object detection learning framework
  - Instead of selecting a priori the complexity of the classifier, defining a criterion for stopping the training of each layer improves the performance of the classifier.
  - It is possible to design a training procedure that is scalable on the number of training examples and the number of features.

  - Sampling features during the boosting process can speed up considerably the training process without affecting considerably the robustness of the obtained classifier.
  - Using internal bootstrapping in the training of a cascade helps to produce more compact cascades/coarse-to-fine classifiers and increases the detection rates.

- Multiclass coarse-to-fine boosted classifier:
  - Boosted cascade classifiers that have been used for one-class object detection problems can be extended to the multiclass case using a coarse-to-fine multiclass classifier. This can be done by training each stage of the coarse-to-fine classifier with multiclass versions of Adaboost (such as JointBoost [Torralba et al., 2007], Adaboost.MH [Schapire and Singer, 1999], Adaboost.MO [Schapire and Singer, 1999], or ADTrees [Freund and Mason, 1999]).
  - Nested cascade classifiers can be extended to multiclass nested coarse-to-fine classifiers.
  - It is possible to design a training procedure that is scalable on the number of training examples, the number of features, and the number of classes.
  - The bootstrapping procedure can be extended to the multiclass case.
  - In the multiclass case, sharing weak classifiers or features between classes will reduce the processing time without affecting significatively the classification accuracy when using a multiclass coarse-to-fine classifier.

2.4.3 Evaluation

The methods/algorithms to be presented will be evaluated using different criteria. These criteria include the error rates trained detections system, the order (big *O*) of the algorithms, the training time, and running time of the classifiers.

Before continuing it is important to mention that when evaluating a detection system, the obtained performance does not only depends on the classifier, but also on the other modules of the systems, including the method used to downscale the images in the multiresolution module, as well as the method used to determine the final detections. Given that we are mainly interested on the performance of the whole detection system, the evaluation will be performed for the detection system and not for the classifier isolated from the system. The main measures to be used are:

- The Detection Rate (*DR*), also called True Positive Rate (*TPR*). This will be measured per class and is defined as the percentage (rate), per class, of detected objects.
- The Number of False Positives (*NFP* or *FP*): Number of background windows detected as an object[h].
- Receiver Operating Characteristic (ROC) curves (*DR* vs *NFP*) will be build using standard datasets whenever available.

[h]Here, as in most of the recently published literature, the *FPR* is calculated for a given database and not as an average or a rate, mainly because it depends on factors such as the size of the images, the number of analyzed windows, and the difficulty of the database.

For evaluating the scalability of the training, the scalability of the detector system, the training time and processing time, the following will be measured a needed:

- The running time of the detector. This will be measured as function of the number of classes in the multiclass case.
- The the training time of the detector as function of the number of classes, number of features and number of training examples.
- The size of the classifier (number of layer of the cascade, number of weak classifier) and size of the coarse-to-fine classifier (average depth, maximal depth, number of nodes).
- The training time of the detector. This will be measured as function of the number of classes in the multiclass case.

The performance of the whole system will be compared with methods published on the literature using standard databases. This kind of comparison is not always straight forward because of differences in the procedures and training databases, but it gives a overall idea of the performance of the system compared to related work.

Experimental Data

As it was previously mentioned, the face detection problem will be used as base problem to evaluate the propose methods. For this, well known databases for face detection problems, including the MIT-CMU and BioID database for evaluating the frontal face detectors, and the CMU rotated set for multiview face detection problem, will be used. In addition, the UIUC profile car detection image database will be used to evaluate the car detectors. Other problems that will be also used for testing the proposed methods are the problems of hand detection, robot detection, and eye localization.

2.5 Structure of the Document

The present document consist of 6 chapters. Chapter 1 briefly outlined the contributions of the present thesis work, and well as the publications and presentations given during this work. The present chapter (Chapter 2) introduced some basic concepts on object detection, gave an overview of the existing methods for object detection, and presented the hypothesis and objectives of the present work. Chapter 3 presents a review of state of related work on one-class and multiclass object detection problems, with special emphasis on coarse-to-fine, cascade and boosted classifiers, which are some of the main concepts used in the present work.

Later, in Chapters 4 and 5, the proposed methods are described. Chapter 4 (based on the article [Verschae et al., 2008b]) presents the proposed learning framework for one-class detection problems using nested cascades of boosted classifiers. In Chapter 5 an extension of the nested cascade classifier (proposed in Chapter 4) to the multiclass case is presented, a coarse-to-fine multiclass nested cascade is proposed, and a nested multiclass coarse-to-fine tree structure (so called *TCAS*), that uses the multiclass nested cascade as building block is proposed. Finally, in Chapter 6, conclusions of this work are given.

Chapter 3

Related work: coarse-to-fine classifier and boosted classifiers for object detection

Contents

3.1 Coarse-To-Fine Classifiers

3.1.1 Boosting and Object Detection

Boosting

Boosting approaches [Freund and Schapire, 1999] are based on the idea of training the same classifier several times, each time on a different training set or on a different distribution of the training set. Each of the obtained classifiers (called weak classifiers or weak hypotheses) are then used to obtain the final classification. Adaboost [Freund and Schapire, 1999] is one of the most popular boosting algorithm, and it has been widely used in different kinds of classification problems because of its simplicity, high performance, and high training speed.

Adaboost

Taking Adaboost as start point, the boosted cascade paradigm has been widely studied and extended in the last few years in the object detection problem, where [Viola and Jones, 2002] [Viola and Jones, 2004] [Wu et al., 2004] [Fröba and Ernst, 2004] [Lin et al., 2004] and [Liu and Shum, 2003] are some of the most important works in this area, with [Wu et al., 2004] achieving one of the best reported results in frontal face detection. Adaboost builds an additive model of the form

$$H(x) = \sum_{t=1}^{T} \hat{h}_t(x) \tag{3.1}$$

by iteratively adding terms to the sum for minimizing an upper bound of the training error, $E[\exp(-yH(x))]$,[a] with $y \in \{-1, 1\}$ the label of the sample x. This is done by focusing on wrongly classified samples at each iteration of the algorithm. Besides the good performance of Adaboost, the main reasons to use it in object detection problems are that it can be used for feature selection, and more importantly it builds an additive model. Thanks to the additivity of the model, when training a coarse-to-fine classifiers (e.g. a cascade), it is possible to control the computational complexity of each stage of the classifier. This can be done, for example, by selecting the number of terms (T) and/or the complexity of the weak classifiers $\hat{h}_t(\cdot)$.

Some other boosting algorithms (mainly variants of Adaboost) that have been used together with coarse-to-fine classifiers include asymmetrical Adaboost [Viola and Jones, 2002], Gentleboost [Lienhart et al., 2003], Floatboost [Li and Zhang, 2004], KL-Boosting [Liu and Shum, 2003], and JS-Boosting [Huang et al., 2005].

Features and Weak Classifiers

In [Viola and Jones, 2004] a particular feature $f_t(\cdot) \in \mathbf{F}$ is associated to each weak classifier:

$$\hat{h}_t(x) = h_t(f_t(x)), h_t \in \mathbf{H} \tag{3.2}$$

where each feature, $f_t(\cdot)$, is selected from a family of features $\mathbf{F}$ consisting of Haar-like wavelets that can be evaluated very efficiently using the so-called integral image. Examples of other features that have been used in similar systems are Edgelets [Wu and Nevatia, 2007] (car and pedestrian detection), modified Local Binary Patterns (mLBP) [Verschae et al., 2008b] [Fröba and Ernst, 2004] (face detection), granular features [Huang et al., 2007] (multiview face detecion), anisotropic Gaussian filters [Meynet et al., 2007] (frontal face detection) and object-part correlation [Torralba et al., 2007] (many objects classes).

Regarding the weak classifiers h_t, families of functions $\mathbf{H}$ that have been used include decision stumps (with binary [Viola and Jones, 2001] [Meynet et al., 2007] and real outputs [Viola and Jones, 2002] [Torralba et al., 2007]), domain partitioning classifiers [Wu et al., 2004] [Wu and Nevatia, 2007] [Verschae et al., 2008b], and CART classifiers [Brubaker et al., 2006] [Brubaker et al., 2008].

The system developed in [Wu et al., 2004] uses Adaboost together with domain-partitioning weak classifiers (originally proposed in [Schapire and Singer, 1999]), which achieves, compared to the weak classifiers used in [Viola and Jones, 2002], a very important improvement in the representation power, but also keeps a simple representation that achieves a good classification accuracy, and reduces the processing and training time.

[a]Note that a classification is correct if and only if the margin, $yH(x)$, is positive: $yH(x) \geq 0$

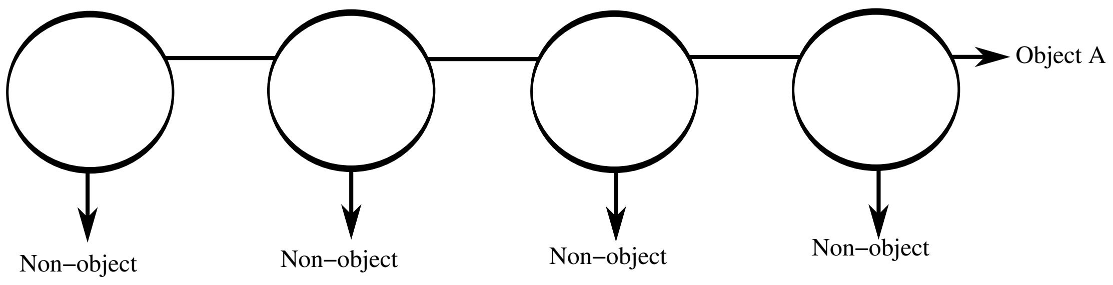


Figure 3.1: Cascade classifier. Example of cascade classifier consisting of 4 layers.

3.1.2 Cascade Classifiers

Cascade boosted approaches were introduced in [Viola and Jones, 2001] for building a face detection system (see an example in Figure 3.1). Cascade classifiers consist of several layers (stages) of classifiers of increasing complexity and are used for obtaining fast processing speed together with high accuracy. This is possible because of two reasons: *a*) there is an important difference in the a-priori probability of occurrence of the classes in any given image, and *b*) most of the non-objects windows are quite different from the object windows. Hence the average processing time of a window is almost completely defined by the expected processing time of non-object windows.

An important improvement over the work on cascades classifiers by [Viola and Jones, 2001] was proposed in [Wu et al., 2004] (and similarly in [Xiao et al., 2003]), where nested cascades are built for reusing information of one stage in the following ones. This is done thanks to the additivity of the model, by letting the classifier used at stage k of the cascade be

$$H_k(x) = H_{k-1}(x) + \sum_{t=1}^{T_k} h_{t,k}(f_{t,k}(x)) \tag{3.3}$$

$$\text{with } H_0(x) = 0 \text{ and } k = \{1, \ldots, K\} \ ,$$

producing faster and more robust cascades compared to the non-nested case. The training of a nested cascade is non-trivial and different procedures have been proposed (see for exampled [Bourdev and Brandt, 2005] [Verschae et al., 2008b]).

Other Cascade Based Classifiers

The literature on cascade classifiers is bast. In the following we outline some relevant and representative methods.

Some works on cascade classifier have proposed to use different classifiers or features at different layers of the cascade. For example, in [Fröba and Ernst, 2004] a cascade consisting of 4 layers is used, where the first 3 layers correspond to boosted classifiers, but the last one is a SNoW classifier [Yang et al., 2000b].

In [Lin et al., 2004], a system based on cascade classifiers that is robust under partial occlusions, is introduced. This system works in the same way as other cascade based systems do [Viola and Jones, 2004], but in case that a possibly occluded face is detected by any layer of the cascade, a parallel, specially designed cascade for that kind of occlusion is evaluated starting from that layer.

An example of a cascade classifier that is not based on a Adaboost is [Schneiderman, 2004], where several cascades are trained to detect not only faces, but also cars, traffic signals, cups, etc. These cascades are based on a so-called feature-centric approach, approach that reuses feature evaluations from adjacent windows without increasing significatively the processing time. The used classifier is a semi-naïve Bayes classifier, and the features are based on a wavelet representation of the image. This method presents, together with [Wu et al., 2004] and [Delakis and Garcia, 2004], the some of best published results for the frontal face detection problem in terms of detection rate and number of false detections; however the obtained classifier is two orders of magnitude slower than [Wu et al., 2004].

3.1.3 Coarse-To-Fine Classifier's Training Procedures

When training a coarse-to-fine classifier (e.g a cascade classifiers) it is not obvious which is the best training strategy, because several interrelated layers are trained at different moments. The training procedure must deal with the following questions: Which examples should be used to train a specific part of the classifier in a given moment? Which kind of feature should be used at a given part of the classifier? And which criteria should be used for stopping the training of a given part of the classifier?

These questions are closely related to the required final performance of the system (TPR, FPR, and processing speed), and to the time required to train the classifier, which in some cases can be quite large (up to months in systems like [Viola and Jones, 2004]). Not many works have taken these questions into consideration.

Bootstrapping

In terms of the selection of the training examples, the commonly used procedure employed for selecting the negative examples is to apply the so called bootstrap procedure [Sung and Poggio, 1998]. The bootstrap procedure consist of, after a given classifier has been trained, to re-train the classifier with an enlarged set of negative examples containing new patterns that the current classifier wrongly classifies. This procedure is usually applied several times in order to obtain a good representation of (the boundary of) the negative class.

In the case of cascade classifiers [Viola and Jones, 2004] [Viola and Jones, 2002] [Viola et al., 2003] [Lienhart et al., 2003] [Wu et al., 2004] [Schneiderman, 2004], it has been used only to collect examples to train the following layer of the cascade. The only exception is [Fröba and Ernst, 2004], where it is briefly mentioned that the bootstrapping procedure was also used to re-train the layers of the cascade. In Chapter 4 we analyse this last procedure and show that it allows to obtain a more representative training set, to reduce the number of features used in each layer, and to improve the detection rates.

Detection Rate, False Positive Rate, and Processing Speed

For obtaining an optimal cascade classifier in terms of processing time, false positive rate (FPR) and true positive rate (TPR), an important issue is how to handle the trade-off between the number of features, the FPR and the TPR of the layer. A few works have dealt with the problem of finding a criterion to stop the training of a given part of the classifier. For example, in [Viola and Jones, 2004] [Viola and Jones, 2002] [Viola et al., 2003] [Lienhart et al., 2003], the complexity of each layer of the cascade is fixed before starting the training.

In [Sun et al., 2004] a cost function to be minimized during the training of each layer is defined. This cost function is a sum of two terms; the first term corresponds to the cascade's TPR, which considers the desired target FPR of the complete cascade. The second term considers the expected number of features to be evaluated up to that layer, multiplied by a parameter used to adjust the trade-off between the two terms. The authors do not specify how to select this parameter. The procedure also needs to set the target FPR of the cascade. In [Brubaker et al., 2008] (first in [Brubaker et al., 2006]), the selection of the number of features and the bias at each layer is also performed by minimizing a cost function, that takes into account the probabilities of achieving the desired TPR and FPR for the cascade and the already trained layers.

Training Time

An important drawback of the system proposed by [Viola and Jones, 2004] is its long training time. It can take up to several months if it is done in a single computer. Few works [Wu et al., 2003b] [Verschae and Ruiz-del-Solar, 2003] [Brubaker et al., 2006] have tried to overcome this problem when training cascade classifiers[b].

In [Wu et al., 2003b] is proposed to replace the feature selection done by Adaboost by a Forward Feature Selection (FFS) procedure. The FFS can be performed before running Adaboost or it can replace it completely. In [Brubaker et al., 2006] FFS is compared to three methods that select a subset of features prior to running Adaboost. One of the methods (RND) performs a random sampling of the features, the second method (RANK) ranks the features by mutual information, and the third method (CMI) ranks the features by conditional mutual information. They show that RND outperforms RANK, and that FFS and CMI have similar performance, both outperforming RND.

[Verschae and Ruiz-del-Solar, 2003] and [Baluja et al., 2004] propose to sample the feature set before each iteration of Adaboost. In this way the training time is reduced proportionally to the percentage of features considered at each iteration, and Adaboost has the chance to select the features from a large set allowing a large diversity of the select features. In Chapter 4 we present an evaluation of this last procedure.

3.2 Multiclass/Multiview Object Detection using Boosted Classifiers

During the last years, several works have tried to deal with the problem of detecting multiple objects classes, but most of them are based on one-class cascade classifiers – binary classifiers – (e.g. [Wu et al., 2004] [Jones and Viola, 2003b] [Li and Zhang, 2004]). Their main problems are that they do not exploit possible similarities among classes or they are not scalable with the number of classes. Only a few works (e.g [Wu and Nevatia, 2007] [Torralba et al., 2007] [Huang et al., 2007]) have developed multiclass detections systems trying to overcome some of these problems. In the following these methods are briefly outlined.

3.2.1 Parallel Cascades

Examples of works using multiples cascades are [Wu et al., 2004] and [Schneiderman, 2004]. On these works, an object-specific cascade is used to detect a particular object class or a particular

[b] [Brubaker et al., 2008], being a paper by the same authors, includes the analysis done in [Wu et al., 2003b] [Brubaker et al., 2006]

view of the object. This has two disadvantages, the processing time increases, being the sum of the running time of the different classifiers, and the number of positives may also increase, because they sum up (if the detections do not overlap). Therefore, having one classifier for each class makes this approach not scalable with the number of classes in cases where several classes of object are to be detected, nor in the case of multiview/multipose detection. Note that [Schneiderman, 2004] uses a common representation –based on the wavelet transform– for all classes, i.e. features evaluations are shared by all classifiers. This allows to compensate the required computational power in the cases when many classes are considered, nevertheless when running several cascades, all classifier evaluations are still needed, and the number of false positives of the different classifiers will accumulate.

3.2.2 Coarse-to-fine Frontal Face Detection

[Gangaputra and Geman, 2006] and [Fleuret and Geman, 2001] build coarse-to-fine classifiers where the "classes" are fine classifications of the position, scale and rotation of the faces. In [Gangaputra and Geman, 2006] is proposed a criterion for deciding whether, at a level of a coarse-to-fine classifier, "*a group of hypotheses should be "retested" (a cascade) versus partitioned into smaller groups ("divide-and-conquer")*". This criterion is based on the so called *cost-to-power* ratio of a classifier [Blanchard and Geman, 2005], which measures the ratio between the cost of applying the classifier (measured as the computational expense of applying the classifier) versus the power of the classifier (measured as the true negative rate of the classifier). Using this criterion they build a detector of frontal faces that has a coarse-to-fine structure that is similar to the one proposed in [Fleuret and Geman, 2001]. The classifier is coarse-to-fine in scale, orientation, position and object/non-object (retesting). Each of the classifiers that constitute this classifier is trained using the binary version of Adaboost [Freund and Schapire, 1999].

3.2.3 View Estimation

In [Jones and Viola, 2003b] a two stages multiview face detection system is proposed. The system first performs an estimation of the view of the faces and afterwards a classifier designed for the predicted view is applied. This work follows the idea of [Rowley et al., 1998] of estimating the rotation of a window assuming it corresponds to a face, and then processing (classifying) the window assuming that rotation. While [Rowley et al., 1998] estimates the rotation and de-rotates each window before feeding it to a neural network classifier built to detect frontal faces, [Jones and Viola, 2003b] estimates the view by means of a decision tree, and afterwards a view-specific cascade is evaluated. In this way the processing time of the classifier is just the processing time of the view estimation (decision tree) plus the processing time of the selected cascade. One of the problems of this kind of approach is that the view estimation must be very accurate in order to deliver the window to the right (view) classifier, and at the same time it must be very fast, as it is always applied to every window being processed.

3.2.4 Pyramid Detector

In [Li and Zhang, 2004], a pyramid of boosted classifiers is used to detect faces in a multiview setting. Each level of the pyramid contains several binary classifiers that make a decision of whether the window corresponds to a face under particular view or to a non-face. Similarly to cascades classifier, if a window is rejected at a given level, it is no longer processed. Only if a window is classified as non-face by all classifiers of the level, no further processing is done to it, otherwise it is feed to the next level of the pyramid where is further processed by all binary classifiers at that level. Given that each level of the pyramid all binary classifiers are evaluated, the system requires many features and classifier evaluations, requiring to evaluate -in further stages of the pyramid- subranges that could have been already discarded in previous stages. For training the binary classifiers a variant of Adabooost called Floatboost is proposed, which is similar to real Adaboost, but it adds a backward search to remove already added weak classifiers.

3.2.5 Clustered Boosted Tree

In [Wu and Nevatia, 2007] a tree of boosted classifiers for multiview and multipose[c] object (pedestrians and cars) detection problems is proposed. Although the core of the training algorithm works on binary classification problems (based on the work of [Wu et al., 2004]), a multi-view/pose tree classifier is built in a recursive fashion, and a subclass partition is learnt. This is done during training by "splitting" the weak classifiers of the nodes of a tree, as well as the training set. The splitting is triggered when a predefined performance is not achieved at the node being trained. The partitioning of the training set is done by the *k*-means clustering algorithm on the feature space of already selected features. The authors use Edgelet features [Wu and Nevatia, 2005], features that encode edge information, and that are designed to be used in car and pedestrian.

3.2.6 WFS-Tree and Vectoboost

In [Huang et al., 2007] a tree is built for dealing with the problem of rotation invariant face detection. This is done by using a multiclass version of Adaboost, called Vectorboost, which is used to train the nodes of the classifier with a tree structure. At each node, a different subset of face-views is tested (up to 5 views) and sibling nodes work on finer view-ranges. The authors propose to use a Width-First-Search (WFS) tree, where WFS refers to that from any node, any number of siblings of that node can be followed. The basic idea of Vectorboost is to assign to each class a target region, defined as the intersection of half spaces, in the object target space. This algorithm has some interesting characteristics, including: *a*) the target regions for the classes may overlap (the classes do not "compete"), *b*) one class may not share a target region with any of the other classes (i.e. it is a "negative" class). The authors also propose the use of granular features and show that when using them the obtained classifiers are more compact (require the evaluation of fewer weak classifiers), and the final classifier is more efficient compared to the use of rectangular features.

3.2.7 Multiclass Detection With Feature and Weak Classifier Sharing

Regarding multiclass object detection, to the best of our knowledge, the first (and only) work that tries to deal with different kinds of objects using an integrated classifier is [Torralba et al., 2007] (other articles of the same authors related to the same work are [Torralba et al., 2004b] and [Torralba et al., 2004a]).

They introduce a multiclass variant of Adaboost called JointBoost, which is used to build multiclass boosted classifiers and it is tested using up to 32 classes, including frontal faces, cars, keyboards, etc. JointBoost works by finding common features and weak classifiers that can be shared

[c]Here view and pose are used a bit loosely, because the classifier does not output the corresponding view/pose, just the presence of the object and the belonging of the window to a learnt sub-class partition.

across classes.They show that when sharing features and weak classifiers, the processing time of the multiclass classifier grows in sublinear time (close to logarithmic) with the number of object classes, and that the required number of training samples per class is considerably smaller than the one required by most methods. Nevertheless, one of the disadvantages of this method is its high computational complexity (exponential training time on the number of classes), which in practice requires the use of heuristics.

[Torralba et al., 2007] also show that Haar-like wavelets features are not the best choice for many object detection problems and they use features based on the correlation of part of the window with object-parts (introduced in [Vidal-Naquet and Ullman, 2003])[d].

Finally, it is important to recall that in [Torralba et al., 2007] neither coarse-to-fine classifiers nor the bootstrapping procedure are used. In addition, although the results presented in [Torralba et al., 2007] are very promising, the processing speed, the detection rates and false positive rates are not good enough for practical applications; for example the presented false positive rates are $\sim 10^{-3}$ for high detection rates (this is good enough only for small images, in practical applications false positive rates over 10^{-6} are needed).

[d]In the present work we use only Haar-like wavelets (rotated and non-rotated) and mLBP features, which have shown good performance on frontal face detection problems, but they may not fit well many other problems. Future work could consider other feature types.

Chapter 4

A Learning framework of nested cascades of boosted classifiers for object detection systems [a]

Abstract

In this chapter a unified learning framework for object detection and classification using nested cascades of boosted classifiers is proposed. The most interesting aspect of this framework is the integration of powerful learning capabilities together with effective training procedures, which allows building detection and classification systems with high accuracy, robustness, processing speed, and training speed. The proposed framework allowed to build state of the art face detection, eyes detection, and gender classification systems, as well as car, hand and robot detections systems.

Keywords: Object detection, boosting, nested cascade classifiers, face detection, eyes detection, gender classification, cascade training, bootstrapping

[a]This chapter is based on the journal article *[Verschae et al., 2008b]*. In addition to the results presented in that work, results on objects other than faces (robots, hands and cars) are included in the present chapter (see Section 4.4.2).

Contents

4.1 Introduction

The aim of the present chapter is to introduce a unified learning framework for object detection and classification using nested cascades of boosted classifiers. The most interesting aspect of this framework is the integration of powerful learning capabilities together with effective training procedures, which allows building detection and classification systems with high: *i*) accuracy (high detection rates with a few false positives), *ii*) robustness (operation in dynamical environments), *iii*) processing speed (real-time), and *iv*) training speed (a few hours in a standard PC).

For a complex learning machine, having powerful learning capabilities without having adequate training procedures and data is useless. Consequently, the training procedures are as important as the learning algorithm. The proposed framework integrates both aspects, and addresses also the computational complexity aspects of the training. Key concepts of this framework are boosting, nested cascade classification, bootstrap training, and the trade-off between the processing speed and the accuracy of each layer during training.

The proposed learning framework corresponds to a nested cascade of boosted classifiers, and it is based on the seminal ideas proposed in [Viola et al., 2003], with the later improvements introduced in [Fröba and Ernst, 2004] and [Wu et al., 2004]. Our most important contributions over previous work are mainly focused on the adequate training of the nested cascade of boosted classifiers, which also allows to considerably reducing the training time. These contributions include the use of internal and external bootstrap for training the layers of the cascade, the use of feature sampling before each iteration of Adaboost, and a procedure for handling the tradeoff between the TPR, the FPR and the number of features in each layer, procedure that does not requires to predefine the number of weak classifiers of each layer of the cascade.

One of the major advantages of our system is its short training time; it takes only a few hours (e.g. it takes less than 15 hours on for the problem of faces detection using a Pentium 4 and a training set of 5,000 positive examples and 5000 negative examples), and the training time grows linearly with the number of training examples and with the number of features. This is possible thanks to: *i*) the use of nested cascades, *ii*) the use of domain-partitioning weak classifiers implemented using LUTs, *iii*) the use of internal bootstraps, *iv*) the use of a criterion to automatically select the number of each weak classifiers at each layer to, *v*) the use of feature sampling, and *vi*) the use of features that can be evaluated very fast. Some of these aspects also allow obtaining high classification speed.

We apply the proposed framework on the problems of face detection, eye detection, hand detection, car detection, robot detection, and face gender classification. This learning framework could be used also for the construction of detection systems for specific objects view/poses (e.g. heads, pedestrians) and for building (few-classes) classification systems (e.g. race, age).

Chapter Structure

This chapter is structured as follows. In Section 4.2 the nested cascade structure are presented. In Section 4.3 the proposed learning framework is presented, with special emphasis in the description of the training issues, where we present a comparative analysis of the critical components of the proposed learning framework. In Section 4.4, a face detection and an eyes detection systems built using this framework are described, and a comparison of the obtained systems with state of the art systems using standard evaluation databases is given. In addition, the performance of a gender classification system built using the same framework, and results on the problems of hand, car and robot detection are presented. Finally, some conclusions of this work are given in Section 4.5.

4.2 Detection Framework

4.2.1 General Organization

The block diagram of the used detection framework was presented in Figure 2.1 (Chapter 2) and described in Section 2.3.1. As explained, the system is based on five main modules and it is designed for being able to detect faces appearing at different scales and positions within the image being analyzed. Now we highlight a few differences.

As explained, the extracted windows can be then pre-processed for obtaining illumination invariance, by using methods like variance normalization [Viola and Jones, 2001] or histogram equalization [Rowley et al., 1998]. In our system, thanks to the use of rectangular features, and in particular mLBP features, no preprocessing is performed, which allows having shorter processing times.

In the Overlapping Detection Processing module, the windows classified as faces are analyzed for determining the size and position of the final detections. In this module the confidence values associated to the detections are used for fusing them: if the number of overlapped windows in a given position is larger than a given threshold thnum, and also if the detection volume [Delakis and Garcia, 2004] of the overlapped face windows in a given position is larger than a threshold th_{vol}, then the postivive windows are considered as a true detection and fused. The detection volume is defined as the sum of all confidences values corresponding to a set of the overlapped windows corresponding to a face. The fusion procedure is described in [Verschae and Ruiz-del-Solar, 2003].

4.2.2 Nested Cascade

The classification module of the detector is implemented using a nested cascade classifier. A nested cascade of boosted classifiers is composed by several integrated (nested) layers, each one containing a boosted classifier. The whole cascade works as a single classifier that integrates the classifiers of every layer. Figure 4.1 shows the structure of a nested classifier. In the following paragraphs our realization of this concept will be explained. A nested cascade C, composed of K layers, is defined as the union of K boosted (strong) classifiers H_C^k. It should be noted that a given classifier corresponds to the nesting (combination) of the previous classifiers. The computation of H_C^k makes use of the already evaluated classifiers. Then, a cascade is represented as a set of classifiers:

$$C = \bigcup_{k=1}^{K} \{H_C^k\} \ , \tag{4.1}$$

with each H_C^k defined by:

$$H_C^k = H_C^{k-1} + \sum_{t=1}^{T_k} h_t^k(x) - b_k \ , \tag{4.2}$$

and with

$$H_C^0 = 0 \ , \tag{4.3}$$

and $h_t^k(x)$ the so-called weak classifiers, T_k the number of weak classifiers in layer k, and b_k a threshold value. Due to the nested configuration of C, its output is given by:

$$O_C(x) = \begin{cases} sign(H_C^q(x)) & (H_C^k(x) \geq 0, k = 1, \ldots q-1) \wedge (H_C^q < 0 \vee q = K) \\ sign(H_C^1(x)) & H_C^1(x) < 0 \end{cases} \tag{4.4}$$

with a confidence value of a positive detection (i.e. with a positive output for all layers) given by:

$$conf_C(x) = H_C^K(x), \text{ with } H_C^k(x) \geq 0, k = 1, \ldots, K. \tag{4.5}$$

4.2.3 Weak Classifiers Design: Domaing Partitioning Weak Classifiers

The weak classifiers are applied over features computed in every pattern to be processed. Each weak classifier has associated a single feature. The weak classifiers are designed after the *domain-partitioning weak hypotheses* paradigm [Schapire and Singer, 1999]. Under this paradigm, the weak classifiers make their predictions based on a partitioning of the domain $\mathbb{X}$ (decision stumps are the simplest example of domain-partitioning classifiers). Each classifier has a value associated with a partition of this domain. The domain is partitioned into disjoint blocks $X_1 \ldots X_J$, which cover all of $\mathbb{X}$, and for which $h(x) = h(x')$ for all $x, x' \in X_j$. Thus, the weak classifiers prediction depends only on which block X_j the given sample (instance) falls into. In our case the weak classifiers are applied over features, therefore each feature domain $\mathbb{F}$ is partitioned into disjoint blocks $F_1, \ldots, F_J$, and a weak classifier $h(\cdot)$ will have an output for each partition block of its associated feature f: $h(f(x)) = c_j$ such that $f(x) \in F_j$.

For each classifier, the value associated to each partition block (c_j), i.e. its output, is calculated for minimizing a loss function of the margin [Schapire and Singer, 1999], which is also a bound of the training error. Given a training set $S = \{(x_i, y_i)\}_{i=1,\ldots,n}$ and their associated weights $w_t(i)_{i=1,\ldots,n}$, this value depends on i) the number of times that the corresponding feature –computed on the

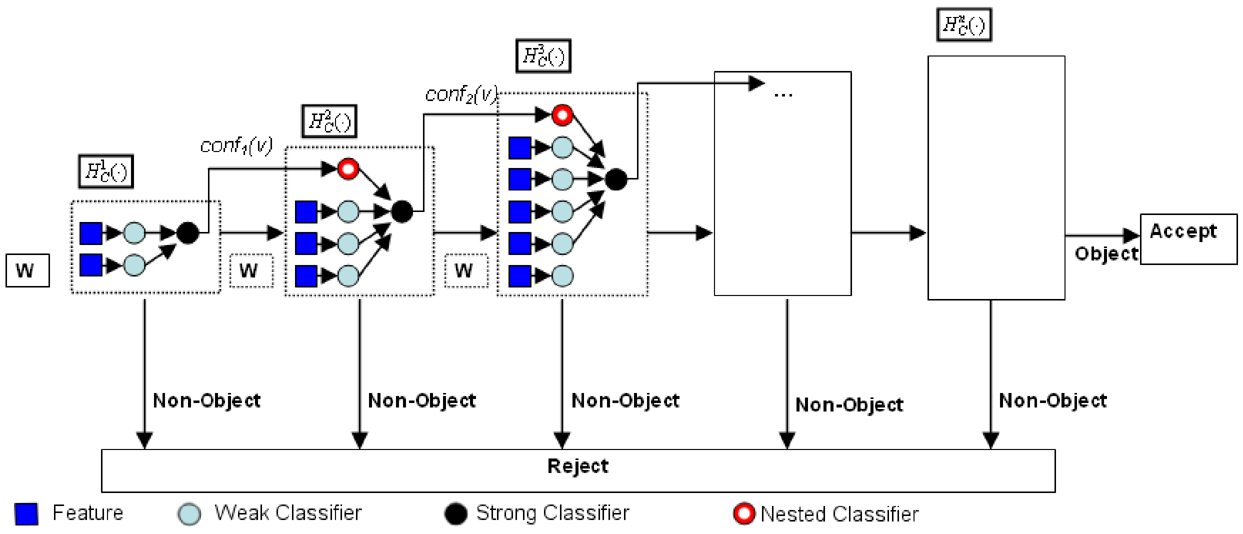


Figure 4.1: Nested Cascade Boosted Classifier

training samples– falls into this partition block (weighted histograms), *ii*) the class of these samples (y_i), and *iii*) their importance $w(i)$. The optimal value of c_j is given by ([Schapire and Singer, 1999]):

$$c_j = \frac{1}{2} \ln \left(\frac{W^j_{+1} + \varepsilon}{W^j_{-1} + \varepsilon} \right), j = 1, \ldots, J \tag{4.6}$$

with

$$W^j_l = \sum_{i: f(x_i) \in F_j \wedge y_i = l} w_t(i) = Pr\left[f(x_i) \in F_j \wedge y_i = l \right] \tag{4.7}$$

with $l \in \{-1, +1\}$ and ε a regularization parameter.

The outputs of a weak classifier (c_j), obtained during training, are stored in a LUT for speeding up its evaluation. Thus, a weak classifier will be defined by $h(f(x)) = h_{LUT}[x] = LUT[index(f(x))]$, with *index* a function that returns the index (block) associated to the feature value $f(x)$ in the LUT. After having evaluated the feature value, this implementation allows –when using equally sized partitions– to compute the weak classifiers in constant time (the same time that takes to evaluate decision stumps) independently of the number of blocks. In addition, the training time of each weak classifier (equation 4.2.3) grows linearly with the number of training examples. Another important advantage of using domain partitioning is the possibility of using features that can not be handled by decision stumps (e.g. mLBP).

4.2.4 Features

The features we employ are rectangular Haar-like features [Viola and Jones, 2001] and mLBP features [Fröba and Ernst, 2004]. Rectangular features are simple features that can be evaluated very quickly, independently of their size, using the integral image [Viola and Jones, 2001]. We partition the range of each rectangular feature in blocks of equal size for obtaining the corresponding weak classifiers. mLBP features correspond to a variant of Local binary patterns LBP (also known as texture numbers or census transform), and unlike rectangular features they are invariant to linear contrast changes. A mLBP feature is obtained by taking a patch of 3x3 pixels within the processing window and comparing each pixel in the patch to the average pixel value of the patch. The output of each of these comparisons is encoded as a 0 or as a 1, and then concatenated for obtaining a binary number of 9 bits. Therefore, for mLBP features the partition of the domain is already defined by the feature itself.

4.3 Cascade Training Framework

When developing a complex learning machine special attention should be given to the training process. It is not only important the adequate selection of the training examples, they should be statistically significant, but also the order in which they are shown to the learning machine. It is also important how to train each part of the learning machine. In the case of a nested cascade of boosted classifiers, it is not obvious which the best training strategy is, because several interrelated layers are trained at different moments. The following questions should be answered: Which examples should be presented to a specific part of the machine in a given moment? Which criterion should be used for stopping the training of a given part of the machine? How are these criteria related with the required final performance of the machine (TPR, FPR, and processing speed)?

4.3.1 Training Procedure: Design of the Strong Classifier

The real Adaboost learning algorithm [Schapire and Singer, 1999] is employed for selecting the features and training the weak classifiers. The implemented real Adaboost learning algorithm takes into account both, the nested configuration of the cascade (a given layer of the cascade should not be trained without taking into account the previously trained layers, see equations (4.2)-(4.4)), and also the asymmetrical distribution of the two classes. The pseudo code of this algorithm is shown in Algorithm 1.

The main idea of cascade classifiers is to process most non-object windows as fast as possible, and to process carefully the object windows and the object-like windows. The final TPR and FPR of the whole classifier are the product of the corresponding rates in each layer. Therefore, for obtaining a high TPR and low FPR for the whole cascade classifier, high TPRs (larger than 0.99) but reasonable low FPRs (lower than 0.5) in each layer are required. For instance, a cascade with 10 layers and TPR and FPR values of 0.999 and 0.2, in each layer, will produce a resulting TPR and FPR of 0.99 (0.999^{10}) and $1.024x10^{-7}$ (0.20^{10}), respectively. Therefore the design and selection of the optimal parameters of each node of the cascade is very important. For handling the tradeoff between the TPR, the FPR, and the processing speed we do not fix the number of features nor the target rates in each layer as most of the works do [Viola and Jones, 2001] [Viola and Jones, 2002] [Fröba and Ernst, 2004] [Wu et al., 2004]. We manage this by setting the maximum allowed FPR (fprMax) and the minimum allowed TPR (tprMin) per layer, while the minimum number of features is selected such that fprMax and tprMin are achieved. At each iteration of Adaboost several values of a bias, $b_k \geq 0$, are tested for fulfilling the classification requirements of the layer (see Algorithms 1 and 3).

As [Sun et al., 2004] we need to select a priori two parameters. The main difference between our procedure and [Wu et al., 2004] is that we minimize the number of features following a greedy procedure that assures the minimum desired TPR for each layer, while [Wu et al., 2004] propose a greedy optimization procedure for maximizing the TPR using a cost function that considers the target FPR and the expected number of evaluated features.

Algorithm 1 TRAINLAYERNESTED($H_C^k, H_C^{k-1}, PT, NT, PV, NV, fprMax, tprMin$)

```
Input: PT/PV: Positive Training/Validation Set
Input: NT/NV: Negative Training/Validation Set
Input: $tprMin$: minimum allowed true positive rate for a layer
Input: $fprMax$: maximum allowed false positive rate for a layer
 1: Training Set $S = NT \cup PT = \{(x_i, y_i)\}_{i=1,...,n}$, with $y_i \in \{-1, +1\}$
 2: $w_1(i) \leftarrow \begin{cases} \frac{1}{|NT|} & x_i \in NT \\ \frac{1}{|PT|} & x_i \in PT \end{cases}, i = 1, \ldots, n$
 3: $w_1(i) \leftarrow w_1(i) \exp(-y_i H_C^{k-1}(x_i))$, $i = 1, \ldots, n$
 4: Normalize $w_1$ to a p.d.f
 5: $H_C^k \leftarrow H_c^{k-1}$
 6: $t \leftarrow 1$
 7: while (!VALIDATECLASSIFIER($H_C^k, k, PV, NV, fprMax, tprMin$)) do
 8:     for each feature $f_p \in \mathbf{F}, p = 1, \ldots, P$ do
 9:         $W_l^j = \sum_{i: f_p(x_i) \in F_j \wedge y_i = l} w_t(i)$   where $l = \pm, j = 1, \ldots, J$
            $\hat{h}_p(j) = \frac{1}{2} \ln \left( \frac{W_{+1}^j + \varepsilon}{W_{-1}^j + \varepsilon} \right)$
            $Z_p = 2 \sum_j \sqrt{W_{+1}^j W_{-1}^j}$
10:     end for
11:     select $h_t$ that minimizes:
            $\min_{p=1,...,P} Z_p$
12:     $H_C^k(\cdot) \leftarrow H_C^k(\cdot) + h_t(\cdot)$
13:     $w_{t+1}(i) \leftarrow w_t(i) \exp(y_i h_t(x_i)), i = 1, \ldots, n$
14:     $t \leftarrow t + 1$
15: end while
Output: Trained layer
            $H(\cdot) = \sum_{t=0}^{T} h_t(\cdot)$
```

4.3.2 Selection of the Training Examples: Internal and External Bootstrapping

How to select adequate negative examples when training a specific part of a nested cascade of boosted classifiers is not obvious. Moreover, detection problems require discriminative analysis between objects and non-objects (the rest of the world). This produces two types of asymmetries: *a*) there is a high asymmetry in the a-priori probability of occurrence of the classes: in an image there are much more non-object windows than object windows, and *b*) one of the classes is the negation of the other, therefore there is a high asymmetry in the “size” of the classes (as regions of

Algorithm 2 BOOTSTRAPP(H, NB, S_B)

```
Input: $H$: Classifier
Input: $NB$: Number of negative examples to be collected
Input: $S_B$: Bootstrapp set
1: $S \leftarrow \{\emptyset\}$
2: while $|S| < NB$ do
3:     $x^* \leftarrow$ sample $S_B$
4:     if $H(x) > 0$ then
5:         $S \leftarrow S \cup \{x^*\}$
6:     end if
7: end while
Output: $S$: Negative training set
```

Algorithm 3 VALIDATECLASSIFIER$(H_C^k, k, PV, NV, fprMax, tprMin)$

```
Input: $H_C^k$: Classifier
Input: $k$: current layer index
Input: $PV$: Positive validation set
Input: $NV$: Negative validation set
Input: $fprMax$: Maximum false positive rate per layer
Input: $tprMin$: Minimum true positive rate per layer
1: for $b_l \in B = \{b_1, \ldots, b_L\}$ do
2:     $\hat{H} \leftarrow H_C^k - b_l$
3:     if $fpr(\hat{H}, NV) \leq \prod_{l=1}^{k} fprMAX \wedge tpr(\hat{H}, PV) \geq \prod_{l=1}^{k} tprMin$ then
4:         $H_C^k \leftarrow \hat{H}$
5:         return TRUE
6:     end if
7: end for
8: return FALSE
Output: $H_C^k$
    $tpr(H, P)$: true positive rate of a classifier $H$ evaluated on the set $P$
    $fpr(H, N)$: false positive rate of a classifier $H$ evaluated on the set $N$
```

Algorithm 4 NESTEDCASCADETRAINING()

```
1:  $C \leftarrow \{\emptyset\}$ // Nested Cascade $C$, initially empty
2:  $H_C^0 \leftarrow 0$
3:  $k \leftarrow 0$ // Cascade layer counter
4:  $F_0 \leftarrow 1$
5:  while $F_k > F_C$ do
6:      $k \leftarrow k+1$
7:      $NT_k = \text{BOOTSTRAPP}(C, InitSizeNT, NegIMTrainSet)$
8:      $NV_k = \text{BOOTSTRAPP}(C, SizeNT, NegIMTrainSet)$
9:      for $b = 1, \ldots, NB$ do
10:         $H_k = \text{TRAINLAYERNESTED}(H_C^k, H_C^{k-1}, PT, NT, PV, NV_k, fprMaxL, tprMinL)$
11:         $\hat{C} \leftarrow C \cup H_C^k$
12:         $BNT_k = \text{BOOTSTRAPP}(\hat{C}, \frac{FinalSizeNT - InitSizeNT}{B}, NegIMTrainSet)$
13:         $NT_k \leftarrow NT_k \cup BNT_k$
14:     end for
15:     $C \leftarrow \hat{C}$
16:     $F_k = fpr(C, NV_k)$
17: end while
Output:  Trained Cascade $C$
18: $NB$ : Number of internal bootstrapps
    $PT$ / $PV$ : Positive Training/Validation Set
    $NT_k$ /$NV_k$ : Negative Training/Validation Set at layer $k$
    $BN_k$ : Bootstrapped Negative Training Set at layer $k$
    $tprMinL$ : minimum allowed true positive rate of a Layer
    $fpr(C,N)$ : false positive rate of the cascade evaluated on the set $N$
    $fprMaxL$ : maximum allowed false positive rate of a layer
    $fprMaxC$ : target overall false positive rate of the cascade
    $NegIMValSet$ : set of images not containing positives (to be used in the validation set)
    $NegIMTrainSet$ : set of images not containing positives (to be used in the training set)
    $SizeNV$ : size of the bootstrapped validation set of negative windows
    $InitSizeNT$: inicial size of the training set of negative windows
    $FinalSizeNT$ : final size of the training set of negative windows
```

the input space). From several millions examples (the rest of the world), one should select the ones that correctly define the classification boundary (positive class versus negative class). If we take a face detection system as a case study, every window of any size in any image that does not contain a face is a valid non-object training example. Obviously, to include all possible non-face patterns in the training database is not an alternative. For defining such a boundary, non-face patterns that look similar to faces should be selected. This is commonly solved using the bootstrap procedure [Sung and Poggio, 1998], which corresponds to iteratively train the classifier, each time increasing the negative training set by adding examples of the negative class that were incorrectly classified.

When training a cascade classifier the bootstrap can be applied in two different situations: before starting the training of a new layer and for re-training a layer that was just trained. After our experience, it is important to use bootstrap in both situations. When we apply this procedure for a layer already trained, we call it *internal bootstrap*, while when we apply the bootstrap before starting the training of a new layer, we call it *external bootstrap*. The external bootstrap is applied just one time for each layer, before starting its training, while the internal bootstrap can be applied several times during the training of the layer. The bootstrap procedure in both cases is the same (see BOOTSTRAPP, Algorithm 2) with only one difference, before starting an external bootstrap all negative samples collected for the training of the previous layer are discarded. The use of internal bootstrap in the training of the layers of a cascade classifier is a key point that until now, to our knowledge, has not been analyzed, being just briefly mentioned in [Fröba and Ernst, 2004].

The training procedure of the whole nested cascade is described in Algorithm 4. The nested cascade is trained until the target overall FPR is achieved. The training of each layer includes the use of one external bootstrap (applied before training), and B internal bootstraps (applied after training). Every time an internal bootstrap is applied, the layer under training is rebuilt (reset).

4.3.3 Training heuristics

As previously mentioned the training time is an important issue. The training time mainly depends on two factors, the time required to train each layer of the cascade using Adaboost, and the time required to perform the bootstrap. The time required for the bootstrap depends mainly on the computational time required to evaluate the cascade, which becomes larger for the last layers of the cascade. This happens because more windows need to be analyzed for collecting the negative examples, and because these windows are processed by more layers of the cascade. Therefore the only way to reduce the time required for the bootstrap is having a faster cascade, which we achieve thanks to the use of a nested cascade, and fast weak classifiers (such as the domain-partitioning classifiers previously described). Concerning the training of the layers of the cascade, in the work of [Viola and Jones, 2002] the training time for one layer is $O(n\log(n)|F|T)$ plus the time required for the bootstrap, with n the number of training examples of that layer, $|F|$ the number of features being tested, and T the number of selected features in the layer. The $O(n\log n)$ factor comes from the selection of the threshold of the decision stump. This time can be reduced to $O(n)$ if an "off-line" evaluation of the features for each training example and a sorting of these values is done, but the memory requirement is not longer $O(n)$, but $O(n|F|)$, which can be prohibitive[a] when the training set and number of features are large.

In the present work the training time of the layers is greatly reduced thanks to several factors. First, the weak classifiers can be trained in time $O(n)$ with a memory requirement of $O(J)$, thanks to

[a]With current computers' memory this is becoming no longer valid (approximatly 4 GB of memory is needed when 10,000 examples and 100,000 features are used)

the use of domain partitioning with (J) equally-spaced blocks of the feature domain. This reduces the training time of each layer to $O(n|F|T)$. A second factor for reducing the training time is the sampling of the features before each iteration of Adaboost. This reduces the training time to $O(qn|F|T)$, with $0 < q \leq 1$ the percentage of features considered at each iteration. Third, the training time is further reduced by using rectangular features for the first layers of the cascade and mLBP based features for later layers. For example, when using processing windows of 24x24 pixels and 3x3 pixel neighborhoods for evaluating the mLBP features, there are about 135,000 possible rectangular features and only 484 mLBP features. Therefore the selection of features and classifiers using only mLBP features is about 257 times faster.

4.3.4 Analysis of the Training Procedure

In the following an analysis of the different improvements and variations proposed for designing and training cascades of boosted classifiers is presented. In this analysis we compare the following elements: *a*) the use of normal cascades versus nested cascades, *b*) the application of internal-bootstrap, *c*) the use of rectangular and/or mLBP features, *d*) the effect of the maximum FPR per layer, and *e*) the use of features sampling.

The whole analysis presented in this section was carried out in the face detection problem, namely, using face detectors built using the proposed learning framework. The obtained results should be valid for other classification or detection systems (eyes, gender, race, etc.) to be built using the same framework. For carrying out this analysis more than 9,000 images, obtained from different sources such as Internet and family photograph albums, were employed. No single image employed for testing (see Section 4.4) was employed in this analysis. The following image datasets were built:

- PT (Positive Training set): 5,000 positive training examples obtained from several hundreds images,
- PV (Positive Validation set): the mirrored version (flop) of all faces from the PT,
- NIT (Negative Images Training set): 3,500 images not containing faces, used for the bootstrap procedure,
- NIV (Negative Images Validation set): 1,500 images containing non-faces used for the bootstrap during validation,
- NT_k (Negative Training set for layer k): the negative non-face examples employed for the training of layer k, and obtained using bootstrap from NIT. Due to the use of internal and external bootstrap, this set changes in each layer and in each iteration,
- NV_k (Negative Validation set for layer k): the negative non-face examples employed for the validation of a layer, and obtained using bootstrap from NIV. Due to the use of internal and external bootstrap, this set changes for each layer and in each iteration.

The presented analysis was performed using ROC (Receiver Operating Characteristic) Curves. In these ROCs, each operation point was obtained by evaluating cascade instances with different number of layers, using the validation sets PV and NV. In other words, the parameter to be changed for obtaining the ROCs is the number of layers of the cascades. In the following experiments the minimum TPR per layer was set to 0.999.

Nested Versus Non-nested Cascades

In Figure 4.2(a)[b] is shown the effect of using a nested cascade versus a non-nested cascade in terms of classification accuracy, while in Figure 4.2(b) is shown the number of features obtained for each layer of the cascades. In these graphs it can be seen that in the first layers, the non-nested cascade has larger FPR per layer than the nested one. Moreover, it can be noticed that for the layers 2 to 4 of the cascade, the non-nested cascade needs more than two times the number of features required by the nested cascade. Given that the training and classification time are directly related with the number of features, a nested cascade has a faster training and operation speed than a non-nested one.

Internal Bootstrap

As already explained, we propose to use both internal and external bootstrap. After several experiments we found out that it is better to repeat the internal bootstrap several times, 3 times in the case of our training datasets. However, we wanted to quantify the real effect of this internal bootstrap in the performance of the whole cascade. Figure 4.3(a) shows the effect of performing the internal bootstrap. Clearly the effect for the first layers (the ones with larger FPR) is quite important: the use of internal bootstrap reduces the FPR to the half, while the number of selected features at each layer is almost the same, producing a faster and more accurate cascade.

Feature Sampling During Training

For reducing the training time, at each iteration of the Adaboost feature selection (and weak classifier training) not all features are considered, but only a subset of the possible features [Verschae and Ruiz-del-Solar, 2003] [Baluja et al., 2004]. We have tested the training algorithm considering 100% and 20% (randomly sampled) of the features at each iteration of Adaboost. In Figure 4.3(b) are shown the obtained results. As it can be seen, when using less features better results are obtained for the first layers of the cascade, probably because the chance of over-fitting has been reduced. For the remaining layers of the cascade the performance does not change very much with the number of features. Taking into account this situation, we use only the 20% of the features for reducing the training time.

LBP Versus Rectangular Features

We have tested the use of rectangular and mLBP features. We make three different experiments: training using only mLBP features, training using only rectangular features, and training using both kind of features, rectangular ones for the first two layers and mLBP features for the subsequent layers.

As it can be notice in Figure 4.4(a), the use of only mLBP features has a much lower performance than using only rectangular features, but when using both kind of features, the performance is not greatly affected compared with the situation when only rectangular features are employed. But, why it is interesting to use mLBP features and not just rectangular features? First, mLBP features are invariant to difficult illumination conditions, and they have shown to have a better performance than rectangular features in images with extreme illumination [Fröba and Ernst, 2004].

[b]Notice that in all graphs shown in Figures 4.2 - 4.4, with the exception of Figure 4.2 (b), the operation points shown in the left side (with lower TPR and lower FPR) correspond to later layers of the cascade, while operation points in the right side correspond former layers of the cascade.

Second, the number of mLBP features to be selected is much smaller than the number of rectangular features; therefore, the training is much faster when using mLBP features. Taking this into consideration and also the fact that when using both kinds of features a similar performance is obtained, in our final detection and classification systems we use rectangular features in the first two cascade layers and mLBP features in the subsequent ones.

Selection of Maximum FPR per Layer

The selected maximum FPR per layer has an important effect in the performance of the final system. As it can be noticed in Figure 4.4(b), the use of a larger maximum FPRs per layer reduces the performance of the system instead of increasing it. We think that this is mainly because of two reasons: *a*) the internal bootstrap has a much greater effect when more difficult examples are bootstrapped, which happens when the maximum FPR is smaller, and *b*) large maximum FPRs can induce the selection of too few weak classifiers for a layer, which has a negative effects in the following layers. This effect is also seen in some of the experiments done by [Schapire and Singer, 1999]: at the first iterations of Adaboost the performance is very poor, for boosted classifiers of sizes from 1 to 5-10, the error even increases when adding new weak classifiers, however after adding at least 10 or more classifiers, the error starts to diminished very quickly when new classifiers are added. In our case, when the maximum FPR is large (0.5 in Figure 4.4(b)), only 4 weak classifiers are needed in the first layer, which is a small number of weak classifier for a boosted classifier. On the other hand, when the maximum FPR is lower (0.35 in Figure 4.4(b)), the number of selected weak classifiers in the first layer is 7. We believe that having a very small number of weak classifiers in the first layers can have an important negative effect in final performance of the cascade classifier.

4.4 Evaluation & Results

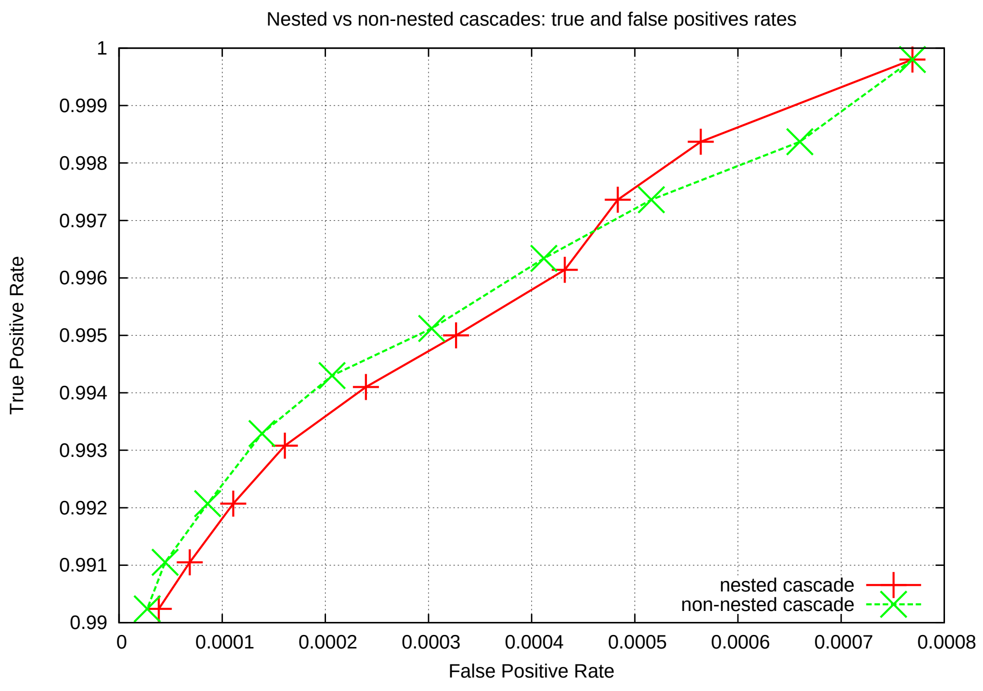
4.4.1 Face Analysis: Face Detection, Eye Localization and Gender Classificacion

Experimental Datasets

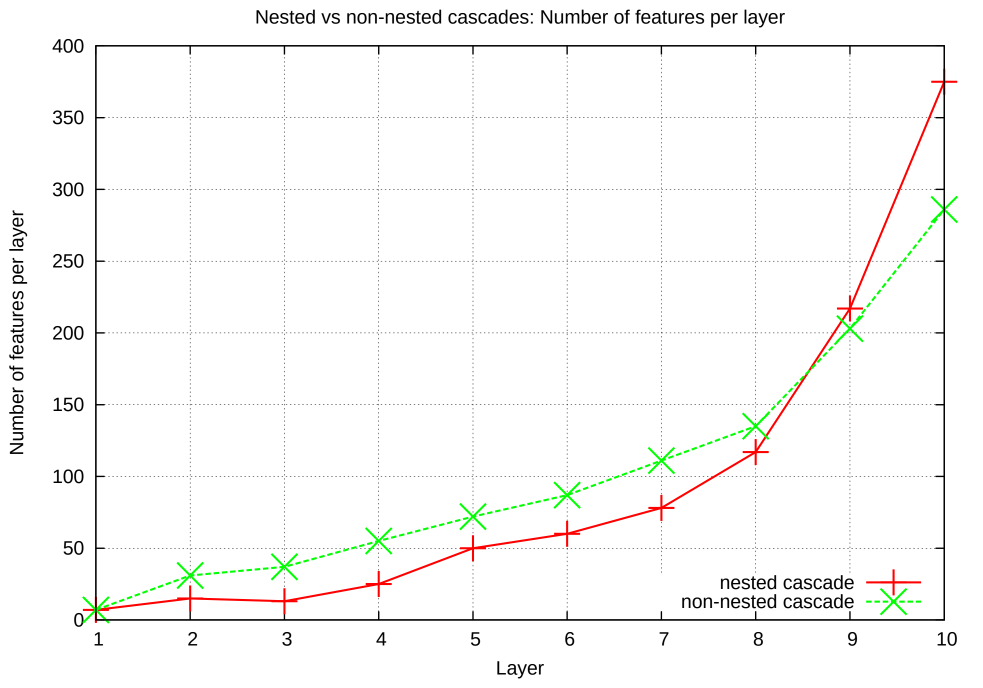
For testing our face and eyes detection systems we employed three standard face databases (BioID, FERET and CMU-MIT), and a new face database (UCHFACE). No single image from these databases was used for the training of our systems.

The BioID Face Database (http://www.humanscan.de/support/downloads/facedb.php) consists of 1,521 gray level images with a resolution of 384x286 pixels. Each one shows the frontal view of a face of one out of 23 different test persons. During the recording special emphasis has been laid on “real world” conditions, therefore the test set features a large variety of illuminations, backgrounds, face sizes, and face expressions.

The FERET database [Phillips et al., 1998] was assembled to support testing and evaluation of face recognition algorithms using standardized tests and procedures. The final corpus consists of 14,051 eight-bit grayscale images of human heads with views ranging from frontal to left and right profiles. For compatibility with our previous study about face recognition algorithms [Ruiz-del-Solar and Navarrete, 2005], we selected 1,016 images containing frontal faces (254 persons, 4 images for each person) for testing our detection systems. The employed FERET subset is available in http://vision.die.uchile.cl.

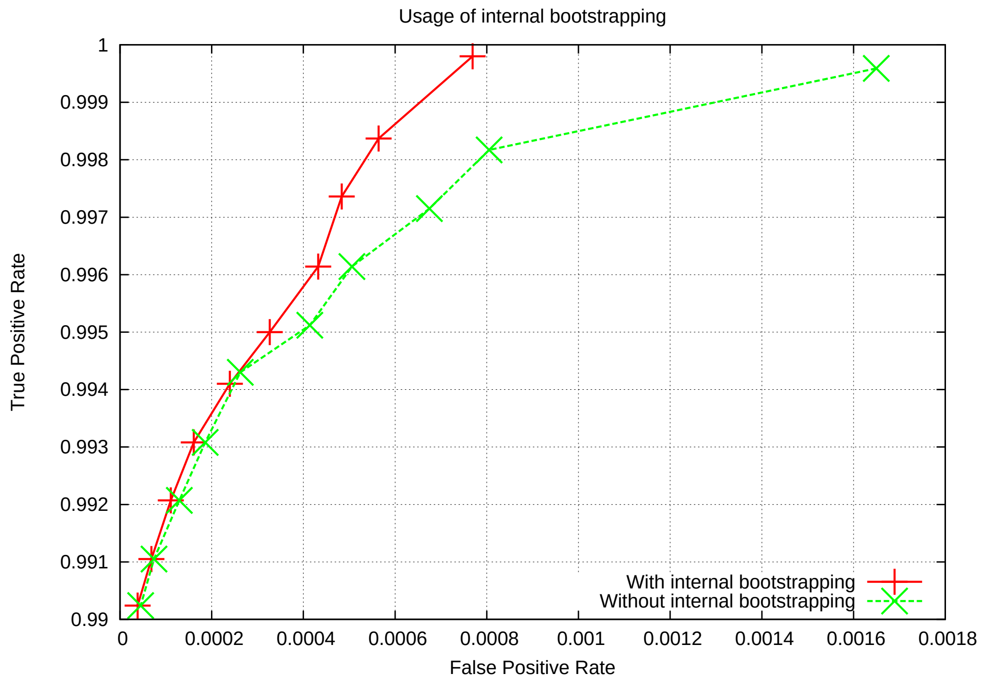


(a)

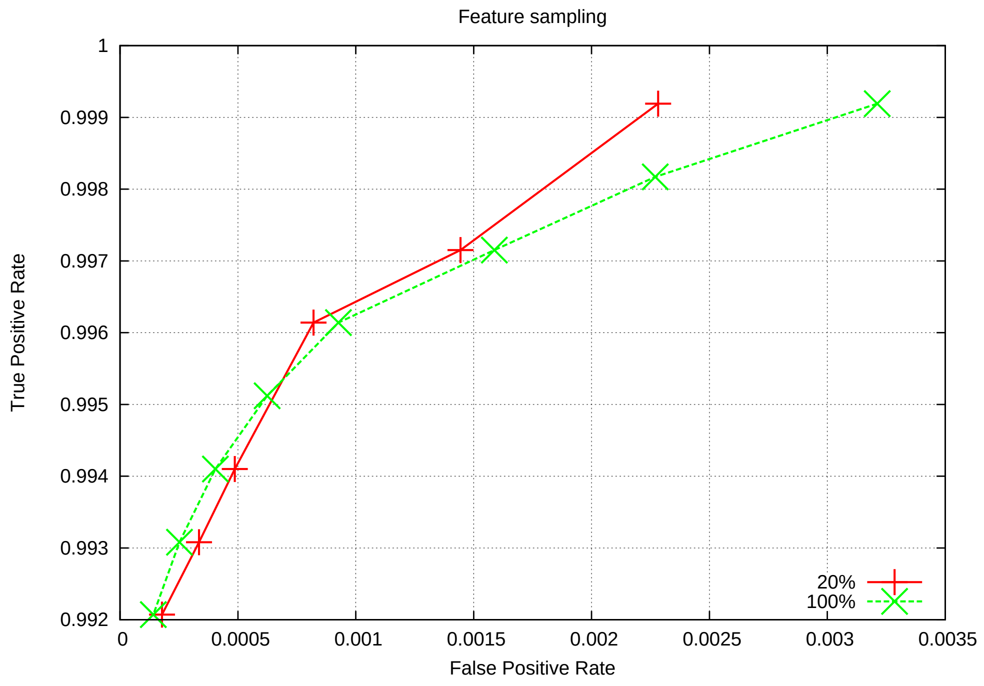


(b)

Figure 4.2: Evaluation of nested and non-nested cascade on the validation set. (a) Positive and negative rates, (b) number of weak classifiersper layer.

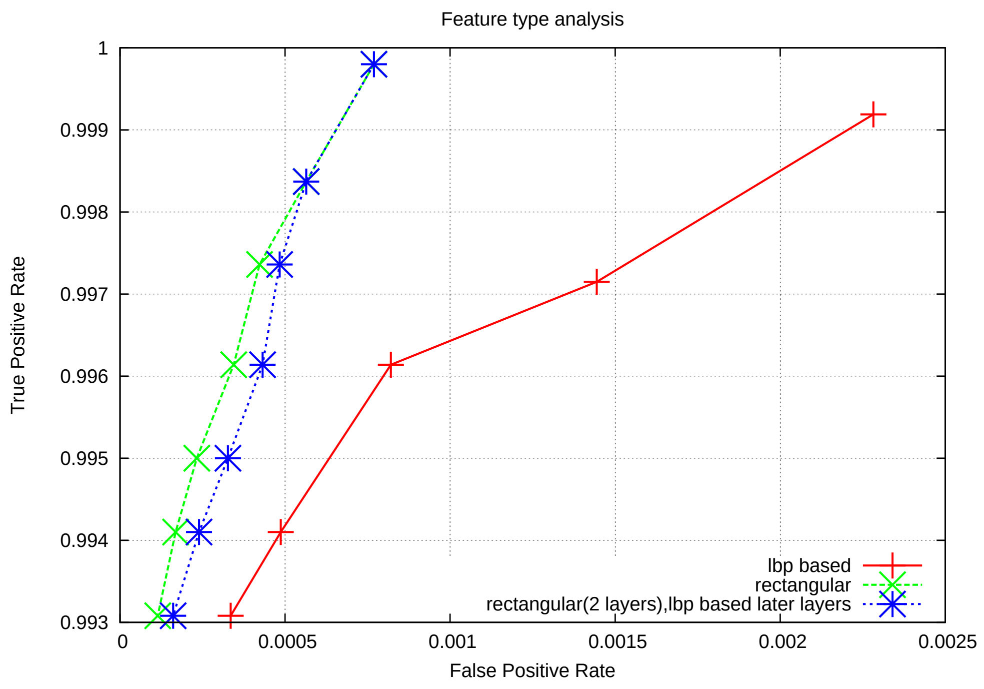


(a)

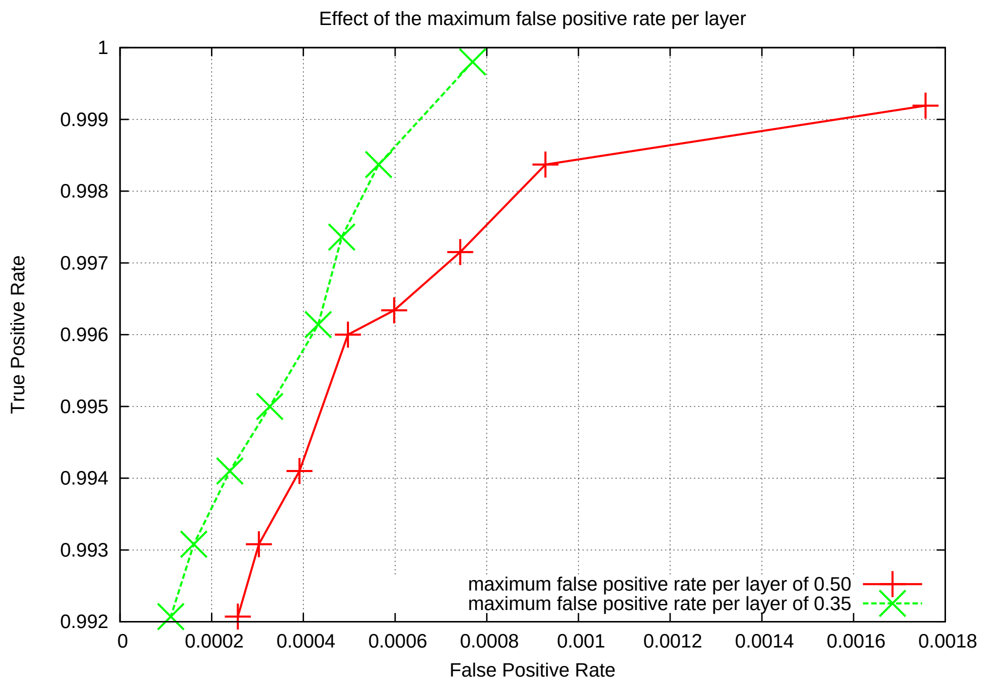


(b)

Figure 4.3: (a) Effect of using internal bootstrapp, (b) effect of feature sampling during training.



(a)



(b)

Figure 4.4: (a) Feature type, (b) effect of selecting different maximum FPR per layer.

The CMU-MIT database [Rowley et al., 1998] consists of 130 grayscale images containing 507 faces. It was originally created for evaluating algorithms for detecting frontal views of human faces. Due to its low resolution and low quality (some images are very noisy), we do not employ it for evaluating the eye detector. It is important to notice that in some publications people has used different subsets of this dataset, because some of the annotated faces are face drawings; therefore comparison is not always straight forward. We use all 130 images and we considered all 507 faces.

The UCHFACE database was especially created for evaluating eyes detection algorithms in images obtained under uncontrolled conditions. It consists of 142 grayscale images obtained from Internet, containing 343 frontal and semi-frontal faces. Face, eyes and gender information was annotated in all these images, and it is available for future studies in http://vision.die.uchile.cl/.

To have a first impression of the capabilities of the built classifiers, in Figure 4.5 are presented some selected examples of face and eyes detection, as well as gender classification, in the FERET, CMU-MIT, and UCHFACE databases.

Algorithms Flavors

Different flavors of the face detection system designed to be used in different types of applications are implemented. In the following paragraphs a brief description of them is given.

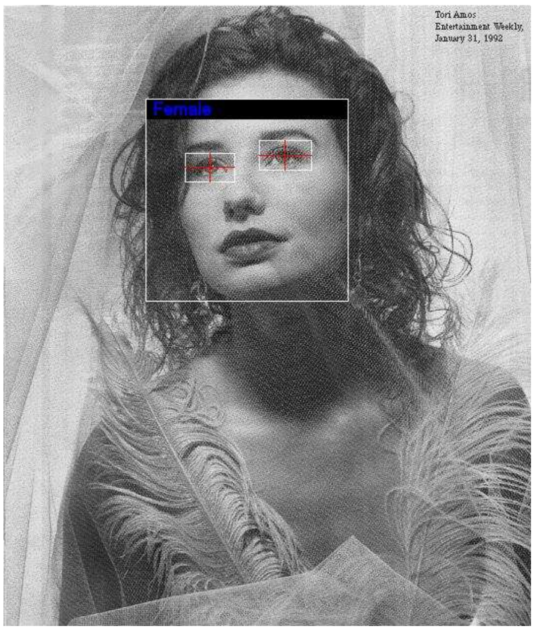
Full search versus Speed search: The difference between Full Search and Speed Search is that in the full search case all possible images windows are considered, while in the speed search a multi-grid approach is employed for selecting the windows positions to be evaluated. The speed search is based in the procedure described in [Fröba and Küblbeck, 2002]. A multi-level grid is defined over the image plane. At the first level of the grid, a step size of 6 pixels is used for selecting the windows positions. At this resolution almost 2.8% of all possible positions are evaluated. At each analyzed grid position the output given by the cascade, i.e. the confidence of the classification, and the index of the last layer where the window was processed, are fed back to the Window Extraction module. At this module it is decided if the image, at neighbor window positions, should be further processed or not. If the confidence is above a threshold ths_1, a second level of the grid is analyzed, considering a finer grid around each starting point of the coarse grid (using a grid step of 3 pixels). Then, each grid position with a score value above a threshold ths_2 is evaluated in the third stage using a 3x3 neighborhood. The speed of the process is controlled by the threshold's values ths_1 and ths_2. Higher threshold's values means higher processing speed, but possible lower detection rates (many true faces can be lost) and false positive rates. Given that a nested cascade is being used, different values of ths_1 and ths_2 for each of the layers have to be selected. This selection is done using a multi-objective genetic algorithm [Verschae et al., 2005], which optimizes the system for having a fast detection speed, a high TPR and a low FPR.

All faces versus Most relevant faces: There are some face detection applications where it is required to detect all possible faces (e.g. surveillance), while in others is required to detect just one (e.g. passport processing) or the most relevant ones (e.g. video conference). For the second case we have implemented a variant of our algorithm that detects just the most relevant faces in a given image or video frame, which reduces considerably the number of false positives. The most relevant faces procedure consists on searching for all faces in a given image, but to filter out the detections that have a confidence value (CV) much lower than the CV of the face with the largest CV. In our implementation much lower means 20 times lower for images where only one face is expected.

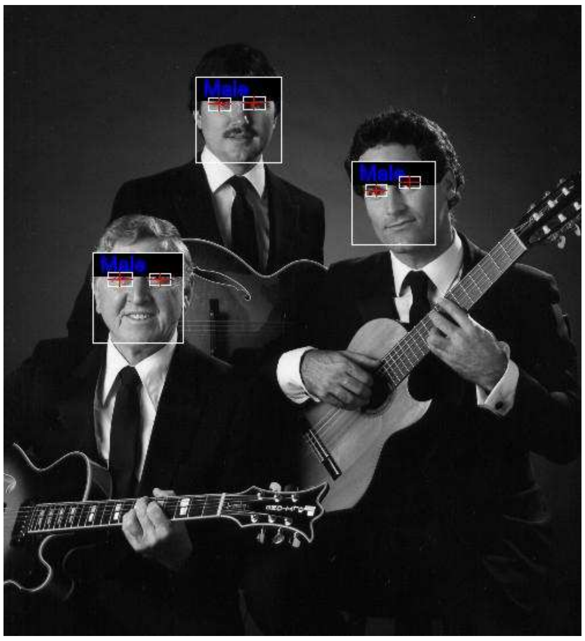
Summarizing, we have four variants of our face detection algorithm: Full-All (full search, all faces), Full-Most (full search, most relevant faces), Speed-All (speed search, all faces) and Speed-Most (speed search, most relevant faces).

Face Detection Evaluation

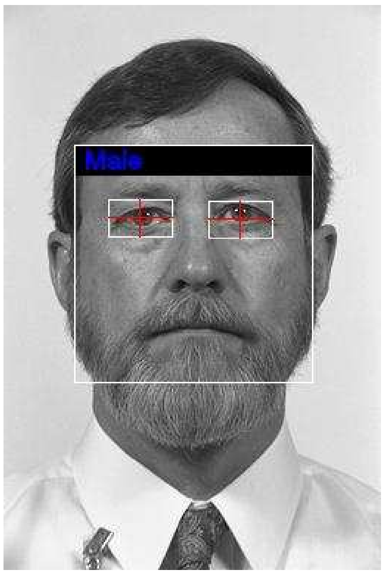
Our face detection system uses a nested cascade composed by domain-partitioning weak classifiers implemented using LUTs. The employed features are rectangular features for the first two layers and mLBP-based for the subsequent layers. The cascade was trained using a maximum FPR per layer of 0.20 (experimentally this value gave us better classification results and a more compact cascade than when using a maximum FPR per layer equal to 0.35), a minimum TPR per layer of 0.999, three internal bootstraps for the training of each layer, and a positive training set of $5,000$ examples. The initial negative training set for each layer had $2,400$ examples (obtained using external bootstrap), and in each internal bootstrap steps 400 more examples were added, obtaining a total of $3,600$ negative examples for the final training of each layer. The final training time of the whole nested cascade was about 15 hours in a 1.8 GHz Pentium 4, with 1,280 MB running Debian GNU/Linux. The obtained final trained cascade has 10 layers.



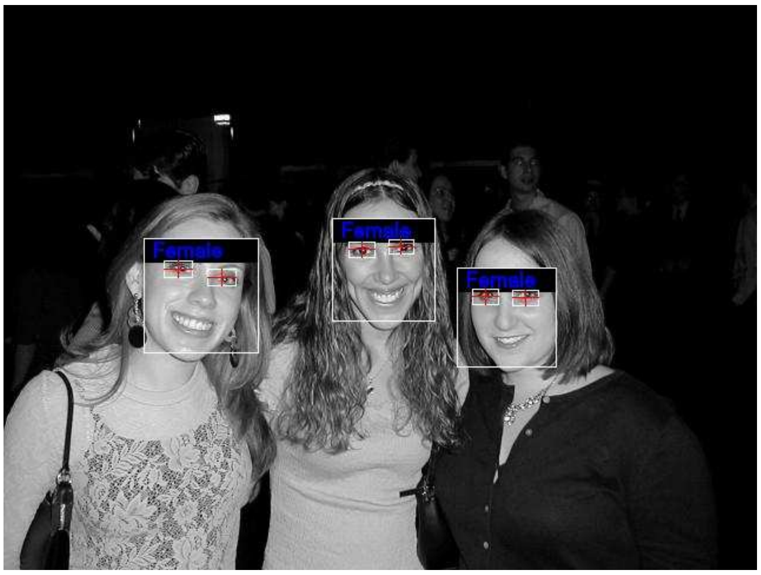
(a) CMU-MIT



(b) UCHFace

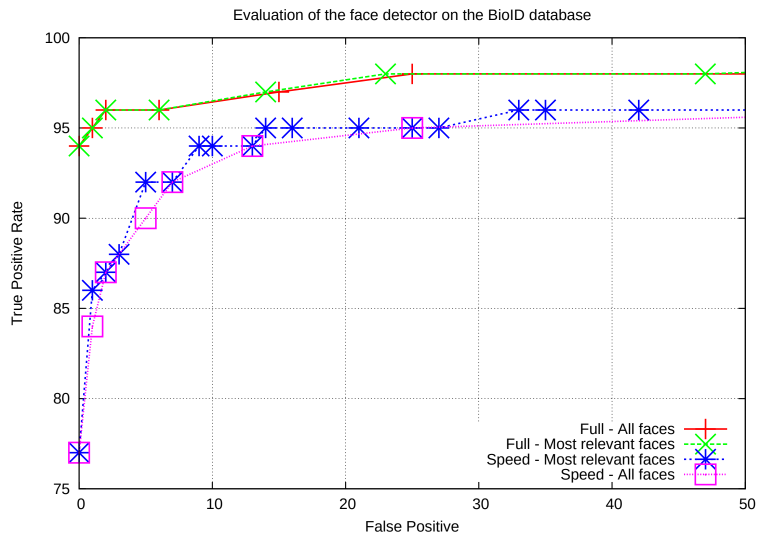


(c) FERET

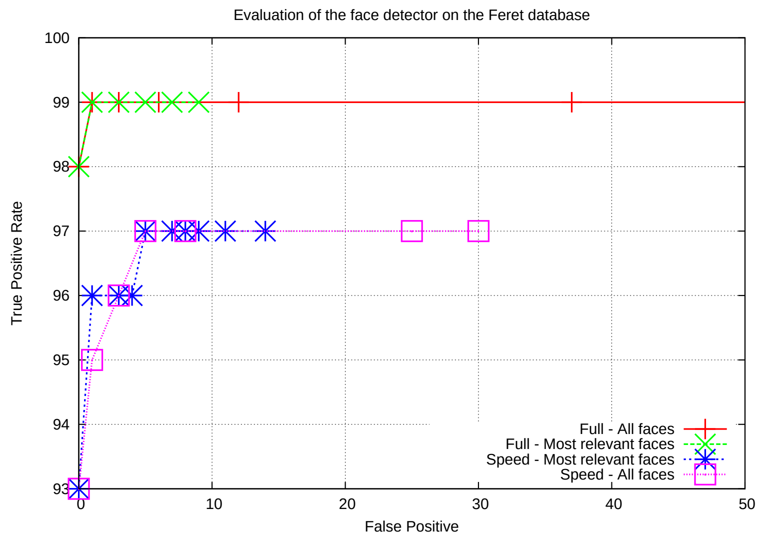


(d) UCHFace

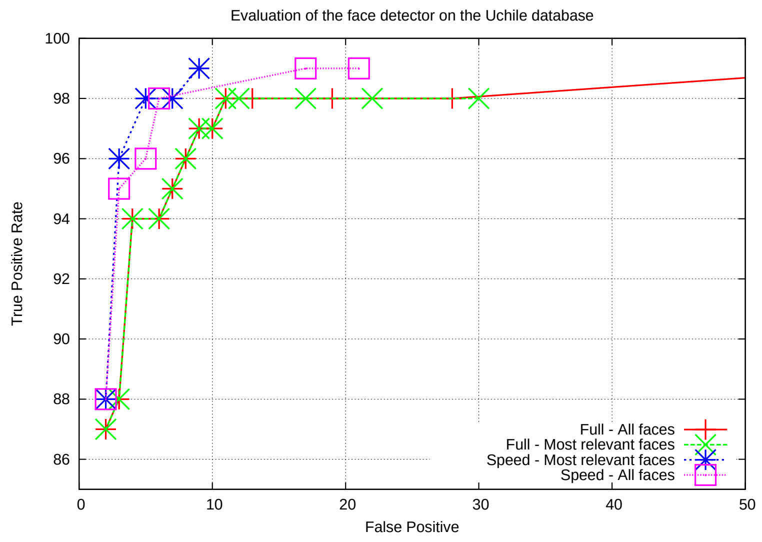
Figure 4.5: Some selected examples of our face detection, eyes detection and gender classification systems at work on standard databases.



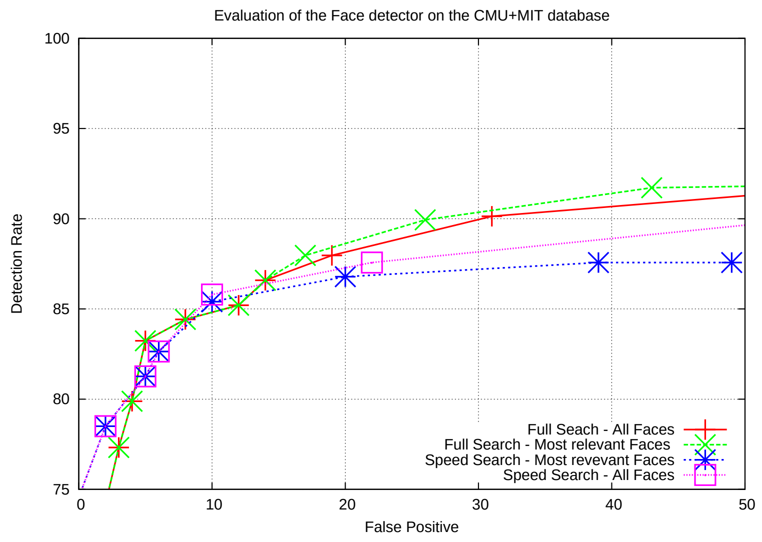
(a) BioID



(b) Feret



(c) UCHFace



(d) CMU-MIT

Figure 4.6: ROC curves of the different face detector flavors in standard databases.

- Face detection in Single Face Images: BioID database. In Figure 4.6 (a) are shown the ROC curves of the different face detector flavors on the BioID database. For this database, selected points of these ROCs are shown in the Table 4.1. In [Fröba and Ernst, 2004] it was reported “while achieving comparable results on the CMU sets, we reach the best published results on the BioID database”. In Table 4.1 it can be seen that our results are much better than to the ones reported by Fröba, especially in the lower parts of the ROCs (low FPR). Other authors reported detection rates of 91.8% [Jesorsky et al., 2001] and 92.8% [Kirchberg et al., 2002], but they do not indicate the FPR. In any case these numbers are lower than ours.

Table 4.1: Comparative evaluation (TPR) of the face detector on the BioID Database (1,521 images)

False Positives	0	1	2	5	6	13	14	15	20	25
Full-All	94.1	95.1	96.5		96.9			97.6		98.1
Full-Most	94.1	95.1	96.5		96.9		97.6			98.1
Speed-All	77.1	84	87.4	90.2		94.6				95.6
Speed-Most	77.1	86.1	88	92.6		94.6	95.1	95.2	95.3	95.6
[Fröba and Ernst, 2004]		~ 50		~ 65				~ 84	~ 94	~ 98

- Face detection in Single Face Images: FERET database. In 4.6 (b) are shown the ROC curves of the different face detector flavors in FERET. Some selected points of these ROCs are shown in Table 4.2. No other groups have reported face detection results in this subset of FERET. Nevertheless, we can affirm that the detection rate in the dataset is very high. From the 1,016 faces to be detected, our best performing algorithms, Full-All and Full-Most, detect 98.7% of them with 0 false positives and 99.5% with 1 false positive. These numbers are very good if we think on the potential application of a face recognition system after the face detection stage (FERET is a face recognition test set).

Table 4.2: Comparative evaluation (TPR) of the face detector on the FERET Database (1,016 images)

False Positives	0	1	3	4	7	8	9	11	12
Full-All	98.7	99.5	99.7						99.7
Full-Most	98.7	99.5	99.6		99.7		99.8		
Speed-All	94	95.7	96.4			97.6			
Speed-Most	94	96.2	96.4	96.7	97.3	97.6	97.7	97.7	

- Face detection in Multiple Face Images: UCHFACE database. In Figure 4.6 (c) are shown the ROC curves of the different face detector flavors on the UCHFACE database. Some selected points of these ROCs are shown in Table 4.3. No other groups have reported face detection results on this new database. However, considering that the images were obtained under uncontrolled conditions, and that they contain several faces per image, we consider that the obtained results are rather good (e.g. 96.5% with 3 false positives, 98.5% with 5 false positives).

- Face detection in Multiple Face Images: CMU-MIT database. In Figure 4.6 (d) are shown the ROC curves of the different face detector flavors applied to the CMU-MIT database. Because of some of these images are noisy and low-quality, all the results here presented were obtained by preprocessing the images with a low pass filter.

Table 4.3: Comparative evaluation (TPR) of the face detector on the UCHFACE (142 images, 343 faces)

False Positives	2	3	5	6	7	8	9	17
Full-All	87.8	88		94.8		96.5	97.1	98.5
Full-Most	87.8	88		94.8	95.9	96.5	97.1	98.5
Speed-All	88.6	95.6	96.8	98.5				99.1
Speed-Most	88.6	96.5	98.5		98.8		99.1	

Table 4.4: Comparative evaluation (TPR) of the face detector on the CMU-MIT database (130 images, 507 faces)
* [Fröba and Ernst, 2004] uses a subset of 483 from 507 faces, set called CMU 125 testset

False Positives	**0**	**3**	**5**	**6**	**10**	**13**	**14**	**19**	**22**	**25**	**29**	**31**	**57**	**65**
Full-All		77.3	83.2				86.6	88		89.9		90.1		92.1
Full-Most		77.3	83.2				86.6						92	
Speed-All	74.6		81.3	82.6	85.8				87.6				90.1	
Speed-Most	74.6		81.3	82.6	85.4			87						
[Fröba and Ernst, 2004] *	∼ 66		∼ 87							∼ 90				
[Wu et al., 2004]		89			90.1	90.7							94.5	
[Viola and Jones, 2001]					76.1							88.4		92
[Rowley et al., 1998]					83.2							86		
[Schneiderman, 2004]				89.7				93.1			94.4			
[Schneiderman and Kanade, 2000]														94.4
[Li and Zhang, 2004]					83.6							90.2		
[Brubaker et al., 2006]				89.1	90.5							93.1		
[Delakis and Garcia, 2004]	88.8				90.5							91.5		92.3

Some selected points of these ROCs are shown in Table 4.4 for comparing these results with the best results reported in the literature in this database. It is important to recall that the evaluation was performed on the same set of images, but all works have used different training sets and procedures, thus a comparisons just gives an overall idea of the performance of the methods, but they can not be directly compared. As it can be seen in Figure 4.6 (d) all flavors have similar performance for low FP (false positives) values. However, for operation points with larger FP, the Full-Most flavor of the face detector gives better results. When we compare these results to the ones obtained by other methodologies presented on the literature in terms of TPR and FP, we obtain better results than [Viola and Jones, 2001] [Rowley et al., 1998], slightly better results than [Li and Zhang, 2004], slightly worse results than [Delakis and Garcia, 2004] (but our system is much faster [c], with a processing time close to 50ms for images of 320x240 on a Pentium 4 computer), and worse results than [Wu et al., 2004], [Schneiderman, 2004] and [Brubaker et al., 2006]. It is difficult to compare our system with [Fröba and Ernst, 2004], because they use a reduced subset of the CMU-MIT databases. But considering that using the complete database is more difficult, because drawings of faces are considered on the original dataset, our results are better than the ones of [Fröba and Ernst, 2004]. We think that we have lower detection rates than [Wu et al., 2004] and [Schneiderman, 2004] mainly because of the size of the training database. As we have mentioned, our training database

[c]The system presented in [Delakis and Garcia, 2004] is about eight times slower [Viola and Jones, 2001] [Viola and Jones, 2002]. Our system has about the same processing time than [Viola and Jones, 2001] [Viola and Jones, 2002].

consists of 5,000 face images, while for example in [Wu et al., 2004] a much larger training set (20,000 faces) was employed. The better performance of [Brubaker et al., 2006] might be because of two reasons: the use of CARTs (Classification And Regression Trees) as weak classifiers, and the criteria used for selecting the trade-off between the FPR and TPR of the classifier. Although [Brubaker et al., 2006] does not give any number, we think that the processing time and training time of our detection system is shorter. One of the reasons is that we use weak classifiers, which can be evaluated just by reading one array (LUT) value – after evaluating the feature–, while [Brubaker et al., 2006] used CART classifiers of depth 4, which require at least 4 comparison evaluations, therefore is at least 4 times slower. In terms of the training time, as already mentioned, in our case it takes $O(n|F|T)$ while the training time of [Brubaker et al., 2006] is at least $O(n\log(n)|F|T)$.

Eye Detection

The eye detector follows the same ideas that the face detector does, i.e. it has its same processing modules. The only difference is that the search for the eyes is performed in the upper part of the face area, i.e. the Window Extraction module extracts windows from already detected face regions. A left eye detector is used to process the left upper part of the detected face, and a right eye detector is used in the same way in the right upper part of the face. Only one eye detector has to be trained (the left eye detector in our case), the other is a mirrored (flopped) version of the one that was trained. Because there are at most two eyes per face, for the eye detector, the Overlapping Detection Processing returns at most one left eye and at most one right eye. The left eye detector we have trained is a boosted classifier consisting of a 1-layer cascade, its weak classifiers are based on rectangular features; it works over windows of 24x24 pixels, and it can process faces of 50x50 pixels or larger. We use only one layer because of two reasons: *i*) the bootstrap procedure is not needed, a representative negative training set can be obtained by sampling not-centered windows of eyes, and *ii*) the processing time is not an important issue because only a small region of the image (the face) needs to be analyzed and the scale of the face is already known. Because the eye detector will be applied to a restricted, reduced image area (upper part of a face), only one flavor of the eye detector is needed, which is equivalent to the Full-Search used for the face detector. The size of the positive training and validation sets was 6,000 in each case, while the size of the negative (non-eye) training and validation sets was 20,000 in each case.

Several eyes detection algorithms have been proposed during the last years. For comparison purposes, we selected state of the art eyes detection algorithms that fulfill the following requirements: *i*) they should be real time, *ii*) a quantitatively characterized of them, using standard databases, should be available, and *iii*) the evaluation databases should include complex background and variable illumination conditions. Two recently proposed systems that fulfill these requirements are [Ma et al., 2004] and [Fasel et al., 2005]. Both of them make use boosted classifiers, are applied after previous stage of face detection, and have been evaluated in the BioID database.

For evaluating the eyes detection accuracy only correctly detected faces are used. We employ cumulative curves of eyes localization relative error [Ma et al., 2004]. The relative error is defined as the Euclidian distance between the ground truth of the eyes and the centers of the detected eyes, normalized by the Euclidian distance between the ground truth eyes centers. We have considered as center of the eyes the middle point between the boundaries of the eye. In Figure 4.7 are shown these cumulative curves for the eye detector on the BioID, FERET, and UCHFACE databases using the different flavors of the face detector. No eyes detection experiments were performed on the MIT-CMU database because in many cases the resolution of the contained faces is too low for performing eyes detection. It can be noticed that all search flavors used for the face detector gives

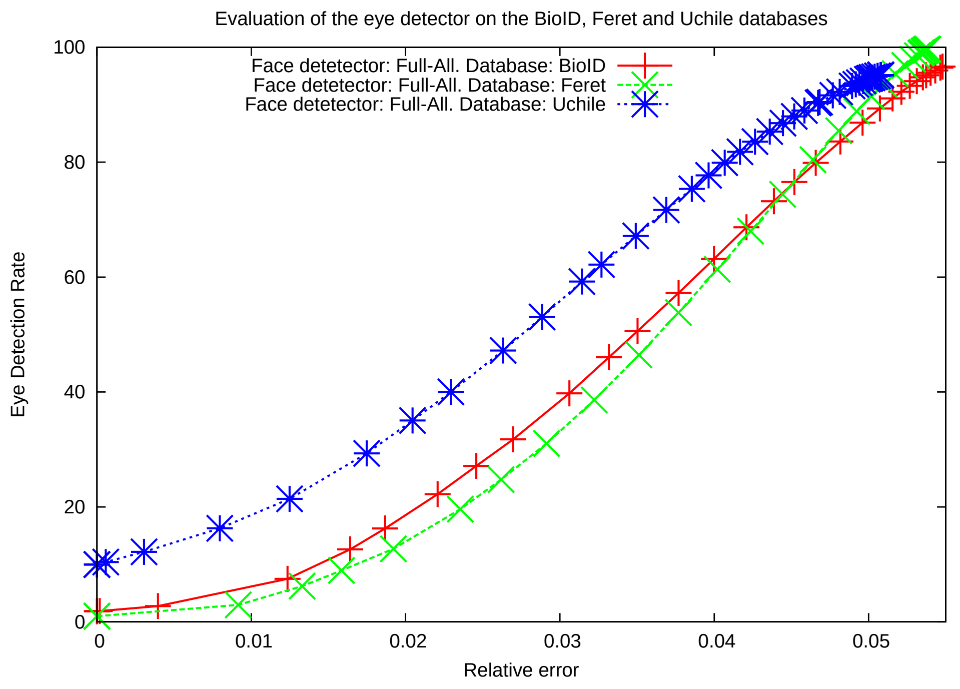


Figure 4.7: Cumulative curves of eye localization error of the eye detector on the BioID , FERET, and UCHFACE databases.

almost the same results for the eye detector. Some selected points of these error cumulative curves, when using the Full-All flavor of the face detector, together with the mean error in pixels for the eyes detection are shown in Table 4.5. The obtained results are very accurate: for example 3.02 pixels error and 97.83% detection rate (DR) in the BioID database. The average distance between the eyes for the BioID database is 54 pixels, hence for a 97.83% detection rate the normalized error (in terms of the eyes distance) is only 5.6%. In the case of the FERET images, for a 99.65% DR the normalized error is 5.38%. In the case of the UCHFACE images, for a 95.16% DR the normalized error is 5.08%.

This small error (close to 5% when all eyes are detected) might be due to an inaccurate annotation of the ground truth (both for the training and evaluation datasets). For example, when annotating a face in which the distance between the eyes is 100 pixels, it is very likely that there will be an error of 5 pixels in the annotation. We think that the obtained accuracy of the detector is very close to the one of a human being, and that it would be very difficult to obtain more accurate results without a very careful annotation of the training faces, and a processing performed at larger faces' resolutions.

By looking at the cumulative error curves on the BioID database we observe that we obtain much better results that the ones reported in [Ma et al., 2004]. For a given eye DR, the error we obtain is less than 50% the one obtained in that work. For instance, for DR of 80% we obtain an error of about 0.047, while in [Ma et al., 2004] the error is 0.1. In [Fasel et al., 2005] a completely different methodology is employed for evaluating the performance of the eye detector. No curves are given but median accuracy measured in terms of irisis. It is very difficult to compare this kind of results with ours. But, by analyzing the employed training methodology, and the eyes detection examples showed in that paper, we believe that those results are comparable with ours. No other groups have reported eyes detection results in our subset of the FERET database or in the new UCHFACE database. We hope in a near future other research groups can employ these databases and the available ground truth for evaluating their systems.

Table 4.5: Comparative evaluation of the eyes detection on the BioID, FERET and UCHFACE databases when the Full-All flavor of the face detector is used. *DR*: Detection Rate, *NE*: Normalized Error, *MEP*: Mean Error in Pixels.

BioID	*DR* [%]	31.8	39.8	50.6	57.2	68.7	79.9	89.4	97.8
	NE	0.027	0.031	0.035	0.038	0.0421	0.047	0.051	0.056
	MEP	1.47	1.67	1.90	2.03	2.27	2.51	2.73	3.02
Feret	*DR* [%]	31.0	38.6	53.8	61.3	68.0	80.4	88.8	99.7
	NE	0.029	0.032	0.038	0.040	0.042	0.046	0.049	0.0548
	MEP	1.84	2.06	2.45	2.65	2.81	3.13	3.35	3.69
UchileDB	*DR* [%]	29.3	40.0	47.2	59.2	71.7	79.9	90.3	95.2
	NE	0.017	0.023	0.026	0.031	0.037	0.041	0.047	0.051
	MEP	0.69	0.86	0.96	1.12	1.29	1.41	1.61	1.71

Gender Classification

Several methods have been proposed for solving the gender classification problem, including neural networks, PCA projections, SVM classifiers, and Adaboost classifiers. Best reported results have been obtained using SVM [Moghaddam and Yang, 2002] [Buchala et al., 2004] and Adaboost [Shakhnarovich et al., 2002] [Wu et al., 2003a]. We will briefly analyze some relevant works. In [Shakhnarovich et al., 2002] a gender classification system based on Adaboost, that uses decision stumps and rectangular features, is presented. The system reached a performance of 79% correct rate in a set of face images obtained from Internet that were manually annotated and cropped prior to the classification. On the same dataset, this system was favorably compared against the one proposed in [Moghaddam and Yang, 2002], being 1,000 times faster and having a higher classification rate (79% against 75.5%). In [Wu et al., 2003a] is described a LUT-based Adaboost system for gender classification that uses rectangular features. Prior to the classification the faces are aligned. This is done through a face alignment method called SDAM that is a kind of AAM (Active Appearance Model). After alignment, grey-level normalization (histogram equalization) is performed. The system achieves a classification rate of 88% on images downloaded form Internet (using 36x36 face windows), and it is favorably compared against a SVM-based system and a decition-stumps based Adaboost system.

We have applied the proposed framework to gender classification using facial information. The implemented gender classification system is applied after face and eyes detection. Face detection is employed for obtaining face windows, which are aligned using the detected eyes and then downscaled to 24x24 windows size. The gender classifier is built using Adaboost classifiers. Implemented features are rectangular and mLBP. Using these features, different flavors of the gender classification system were built. These flavors were evaluated using the FERET, BioID and UCHFACE databases, and compared against SVM-based systems [Verschae et al., 2006]. As can be observed in Table 4.6, when using mLBP features, the proposed gender classification system outperforms SVM-based systems and Adaboost with rectangular features in terms of classification rate. In Table 4.7 it is shown the average time required by the different methods for the gender classification of a given face image. It can be seen that Adaboost-LPB is about 10 times faster than SVM-based systems, while Adaboost-Rectangular is 6 times faster than Adaboost-LPB, and 60 times faster than SVM. To compare with other methods is not easy because the previously used databases are not available. Nevertheless we can observe that the obtained results are similar to the ones obtained by [Shakhnarovich et al., 2002] and slightly lower than [Wu et al., 2003a]. Notice that [Shakhnarovich et al., 2002] does not automatically detect the face and eyes, and that we can handle smaller faces than [Wu et al., 2003a].

Table 4.6: Gender Classification. Correct Classification Rates at Operation points with equal error rates in both classes. Only best performing methods are shown. Faces were cropped using automatically detected eyes.

Database	SVM (RBF)	Adaboost.-Rectangular	Adaboost.-LBP
UCHGender	79.82	79.22	80.12
Feret	84.13	83.95	85.89
BioID	79.05	79.52	81.46

Table 4.7: Average gender classification processing time on a given face image. This time does not include the time required for face detection, face scaling and eyes detection.

Method	SVM	PCA	SVM+PCA	Adaboost-Rectangular	Adaboost-LBP
Time [ms]	10.48	625	205	0.244	1.465

4.4.2 Profile Car Detection, Humanoid and Aibo Robots Detection, and Hand Detection

In the following subsections results on profile car, humanoid robot, Aibo robot, and hands are presented. With the exception of the profile car detection problem, the results are only briefly outlined because for those problems there are no standard databases (results are presented for new databases). These results are presented in order to show how does the proposed framework deal with a wider range of object types.

Car Detection Results

Several method have been proposed for the problem of car detection, including detectors based on a sparse representation [Agarwal et al., 2004], biologically inspired methods [Mutch and Lowe, 2006], Cluster Boosted Trees (CBT) [Wu and Nevatia, 2007], appearance models based on combined categorization and segmentation [Leibe et al., 2004] and unsupervised learning of a category [Todorovic and Ahuja, 2006].

Note that in [Wu and Nevatia, 2007] the use of a Cluster Boosted Tree (CBT) is compared to the use of a cascade detector that has some similarities with the cascade classifier proposed here. Besides the differences on the training procedures, the main difference between their cascade and ours, is that they make use of Edgelets features [Wu and Nevatia, 2005], features that have produced better results compared to rectangular features on the problem of pedestrians detection, and that seem to be better fitted to the problem of car detection.

We have evaluated the proposed learning framework on the problem of profile car detection making use of the test and training sets of the UIUC profile car database [Agarwal et al., 2004]. To train the detector, the UIUC training set was used, set that consist of just 550 cars of low resolution, where the car images were already scaled and cropped to windows of size 100x40 pixels. The training was done using the 550 images downscaled to windows of size 48x24 pixels[d] and no other processing was applied. As validation set, the same images were used after being mirrored.

The settings of the learning algorithm were: a minimum detection rate per layer of 1.0, a maximum false positive rate per layer of 0.35. The used features were rectangular-based and mLBP-based, considering each type of feature on the training of every other layer. In addition, the rotated variant of the rectangular features proposed in [Lienhart et al., 2003] was also used. Given that the training set was rather small, only 64 partitions (bins) were considered for training the domain-partitioning weak classifiers that use rectangular features. As a result, the obtained cascade consist of 9 layers. The obtained detector takes on average 0.35 seconds on a image of size 352x153 (on an Intel Core 2 Duo 2.2Ghz computer), and almost 0.2 seconds when taking advantage of the two cores of the CPU.

The detector was evaluated using the UIUC single-scale and the UIUC multi-scale sets [Agarwal et al., 2004]. The UIUC single-scale set consist of 170 images containing 200 cars of roughly the same scale as the original training images. The UIUC multi-scale set consist of 108 images containing 139 cars at various scales.

Figure 4.8 presents the ROC curves for the profile car detector on these datasets. Examples of detection results are shown in Figure 4.9. To obtain these results the multi-resolution search was done using a scaling factor of 1.15. In order to be able to detect cars that were too close to the image border or partly outside the image, the size of the image was increased by adding to the image a

[d]Unlike other experiments presented in the present work (e.g. faces and hands) in this case the windows are not squared.

black border of 10% of its size. The obtained results are summarized on Tables 4.8 and 4.9. These tables also present results for state of the art methods. These methods only reported results for the operation point at equal precision-recall rates,which corresponds to having equal number of false positives and false negatives.

Now we briefly analyse these results. On the single-scale dataset, the best performing method is [Mutch and Lowe, 2008], follow by [Leibe et al., 2004] and [Wu and Nevatia, 2007] (both, the cluster boosted tree (CBT) and the cascade). Afterwards comes the here proposed method, followed by [Todorovic and Ahuja, 2006] and [Agarwal et al., 2004], in that order. On the multi-scale set, which is a more difficult set, the relative order changes, with the best performing method being [Wu and Nevatia, 2007] (cascade), followed by [Wu and Nevatia, 2007] (CBT), the here proposed method, and [Mutch and Lowe, 2008]. Again, at last come the method proposed in [Agarwal et al., 2004]. In terms of processing time, the cascade and CBT methods [Wu and Nevatia, 2007] show similar results to the cascade proposed here, but all other methods have processing times at least one order of magnitude larger. Compared to the cascade and CBT methods [Wu and Nevatia, 2007], the here proposed method is only slightly worst (3% to 4% for operating points with low false positives). We think this is mainly due to the use different features (Edgelets) in the case of the cascade and CBT classifiers, features that seem to be better fitted for this problem. Note also that a considerably larger training set is used to train the CBT classifier, but on the contrary the detector can also detect cars seen from multiple views.

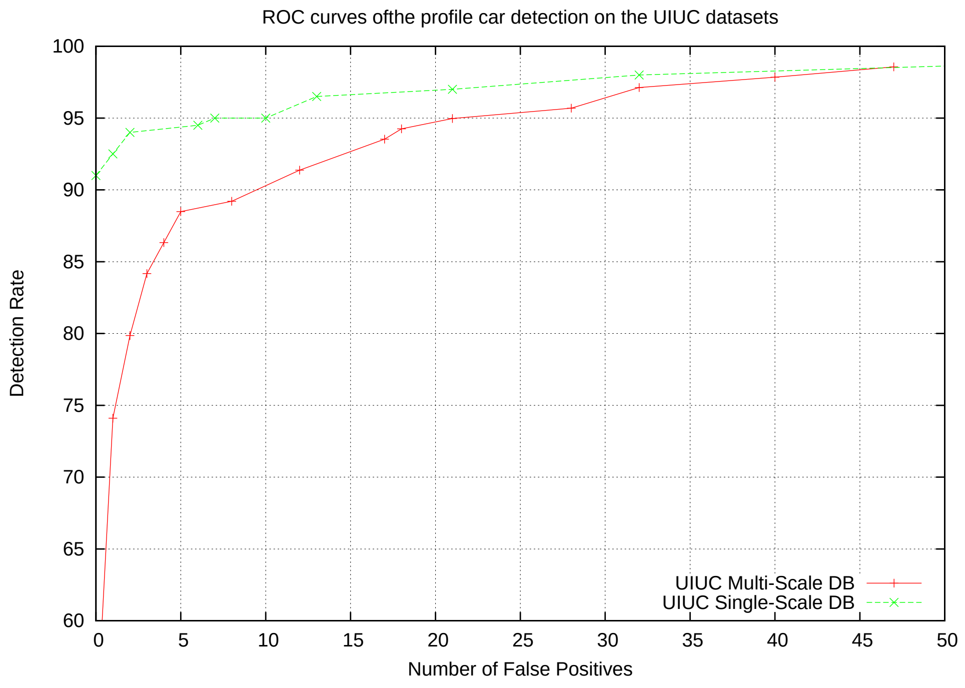


Figure 4.8: ROC curves of the profile detector on the UIUC databases.

Table 4.8: Profile car detection results (FP vs DR [%]) on the UIUC multi-scale database.
* Equal precision-recall rates (equal FP and FN) of the proposed method.

False positives	5	9	10	12 *	13	22	24
[Agarwal et al., 2004] NSA							30.94
[Agarwal et al., 2004] RPE						29.5	
[Mutch and Lowe, 2008]					90.6		
[Wu and Nevatia, 2007] CBT			92.8				
[Wu and Nevatia, 2007] Cascade		93.5					
Proposed Method	88.5	89.9	91.4	91.4	91.4	95.0	93.5

Table 4.9: Profile car detection results (FP vs DR [%]) on the UIUC single-scale database.
* Equal precision-recall rates (equal FP and FN) of the proposed method.

False positives	1	5	7	*10	17	32	33
[Leibe et al., 2004]		97.5					
[Todorovic and Ahuja, 2006]					86.5		
[Mutch and Lowe, 2008]	99.94						
[Agarwal et al., 2004] NSA						70.5	
[Agarwal et al., 2004] RPE							72.5
[Wu and Nevatia, 2007] CBT		97.5					
[Wu and Nevatia, 2007] Cascade		97.5					
Proposed Method	92.5	94.0	95.0	95.0	96.0	98.0	98.0

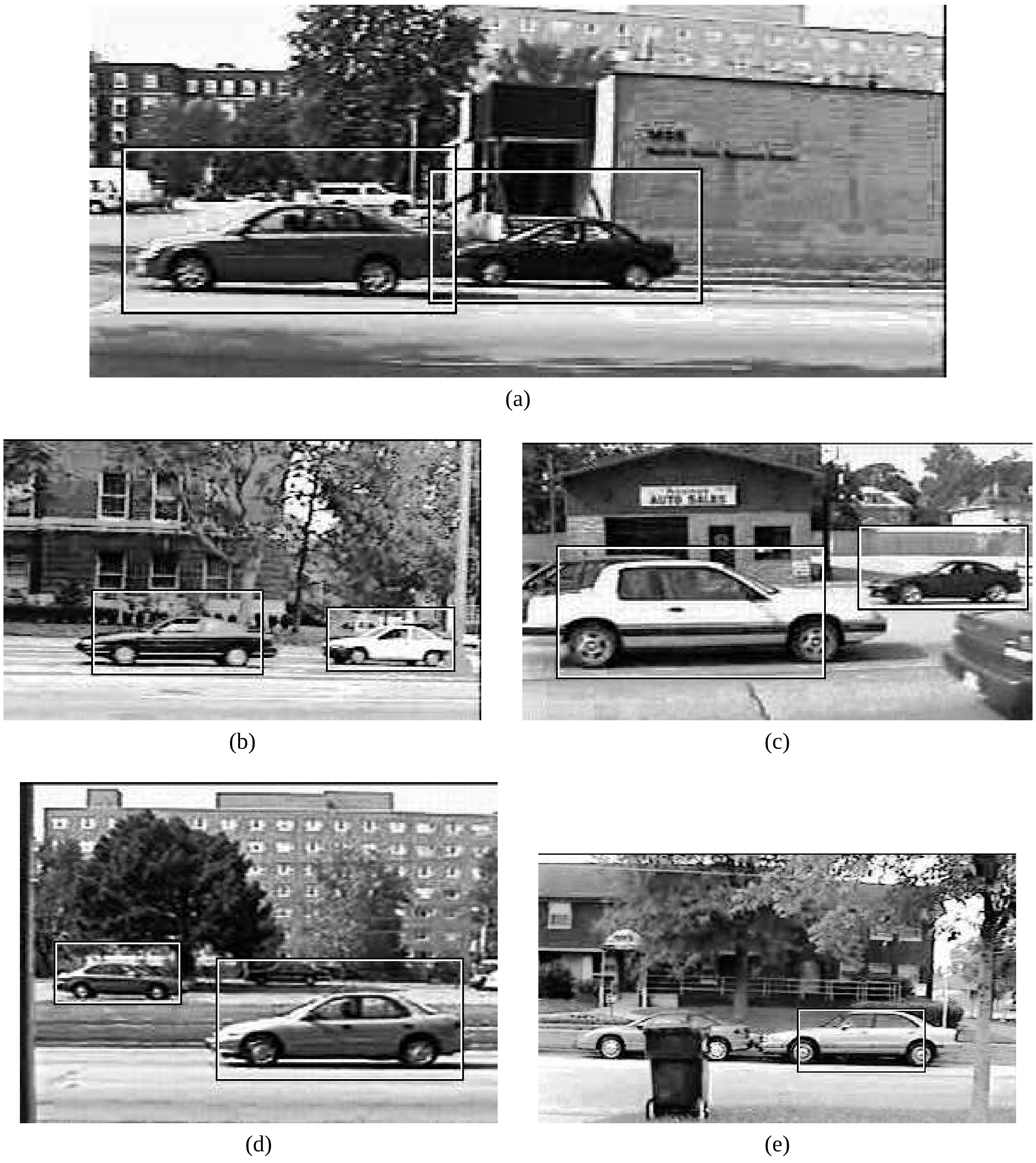


Figure 4.9: Profile car detection results examples on image from the UIUC database. In (c) an occluded car was not detected.

Robot Detection Results

Figures 4.10, and 4.11 present examples of the output of an Aibo robot detector and of a humanoid robot detector. The detectors were trained using the here proposed methods (using the same implementation), but the final system also used additional information, such as the visual horizon as context information (for details see [Arenas et al., 2007] and [Arenas et al., 2009]). Figure 4.12 presents ROC curves of some of the built detectors with and without the use of the visual horizon. Note that the use of the visual horizon is estimated using the camera state of the robot (internal information about the state of the robot's parts), information that reduces considerably the number of false positives (up to a half in some cases).

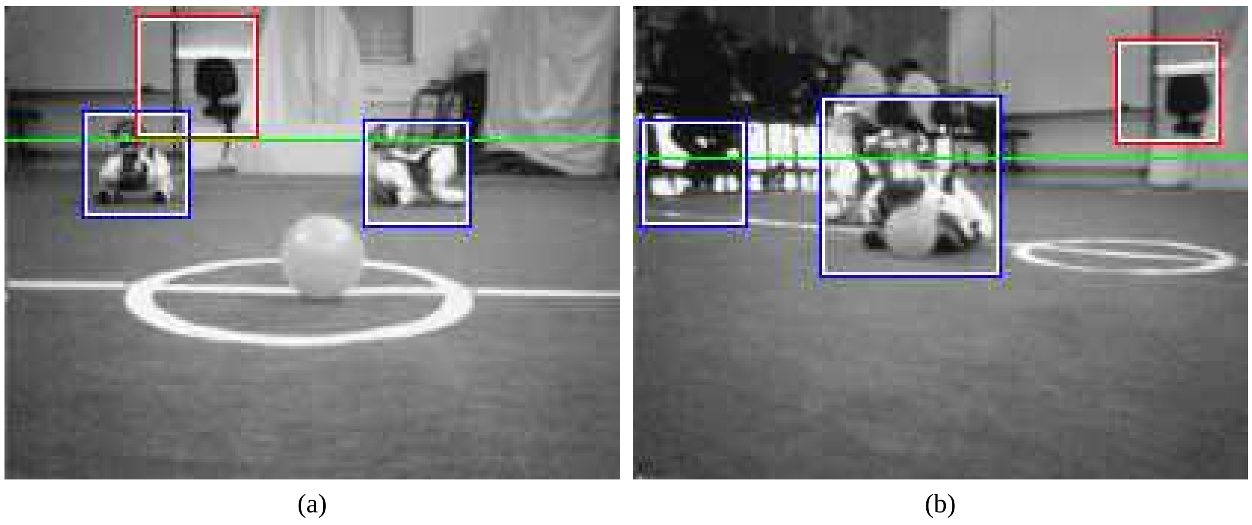
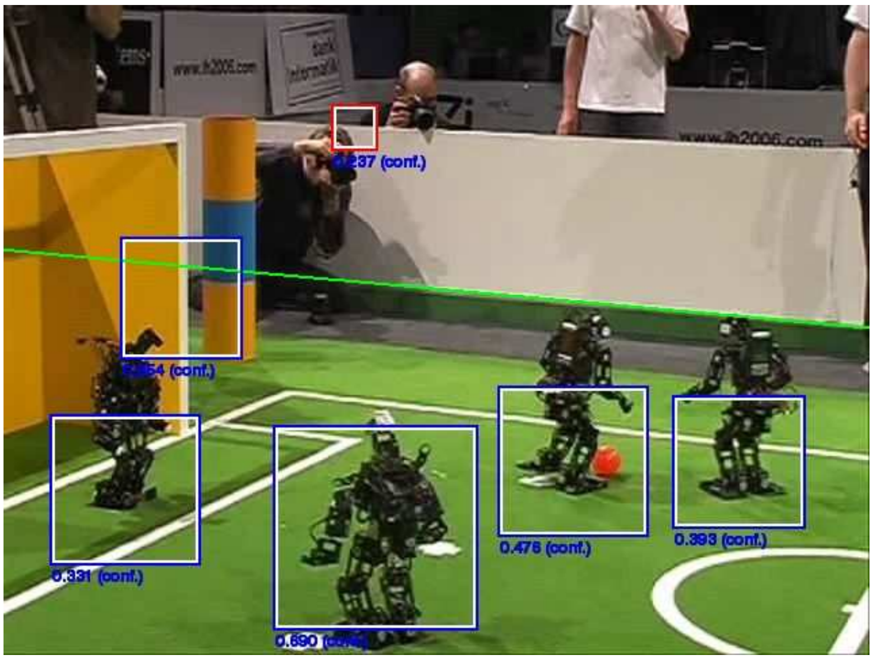
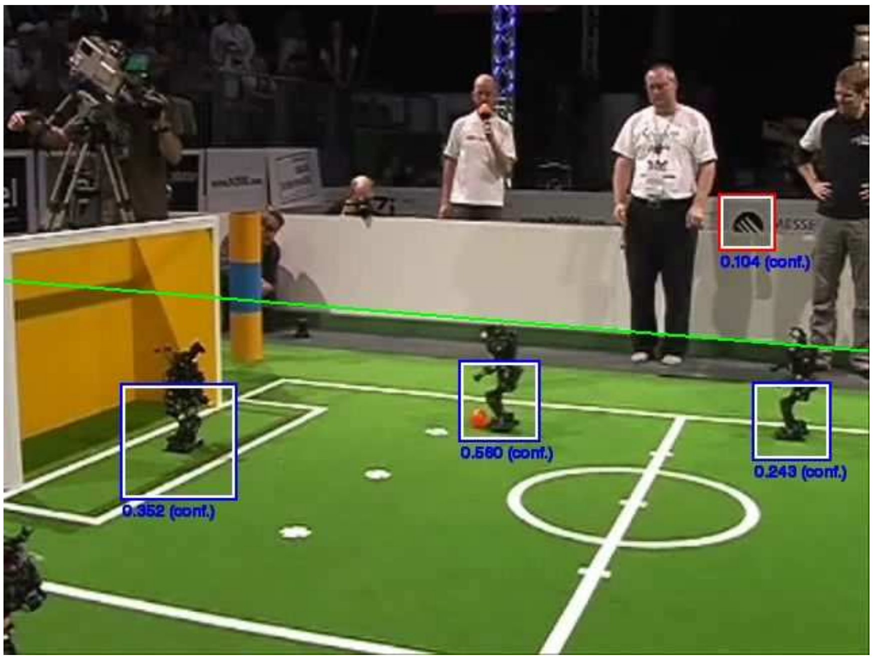


Figure 4.10: Some selected examples of frontal and lateral Aibo Robot detection with and without the use of context information (visual horizon, shown in green). In red are shown the false positives that were filtered thanks to the use of visual horizon.

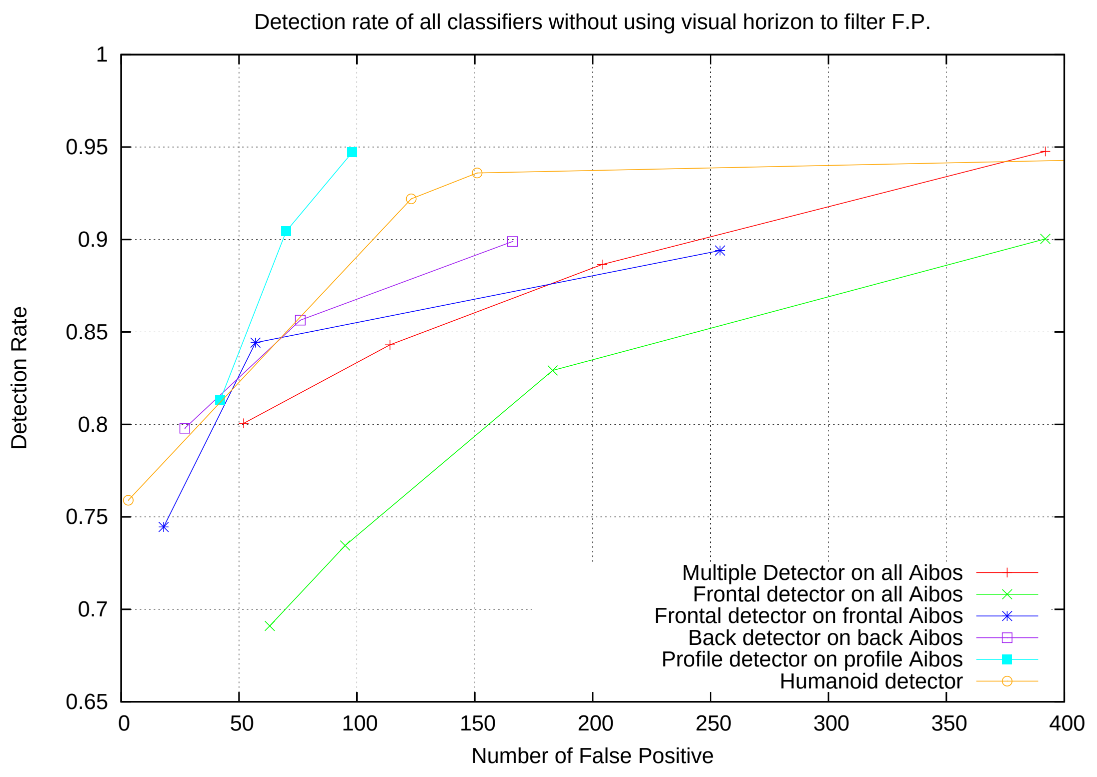


(a)

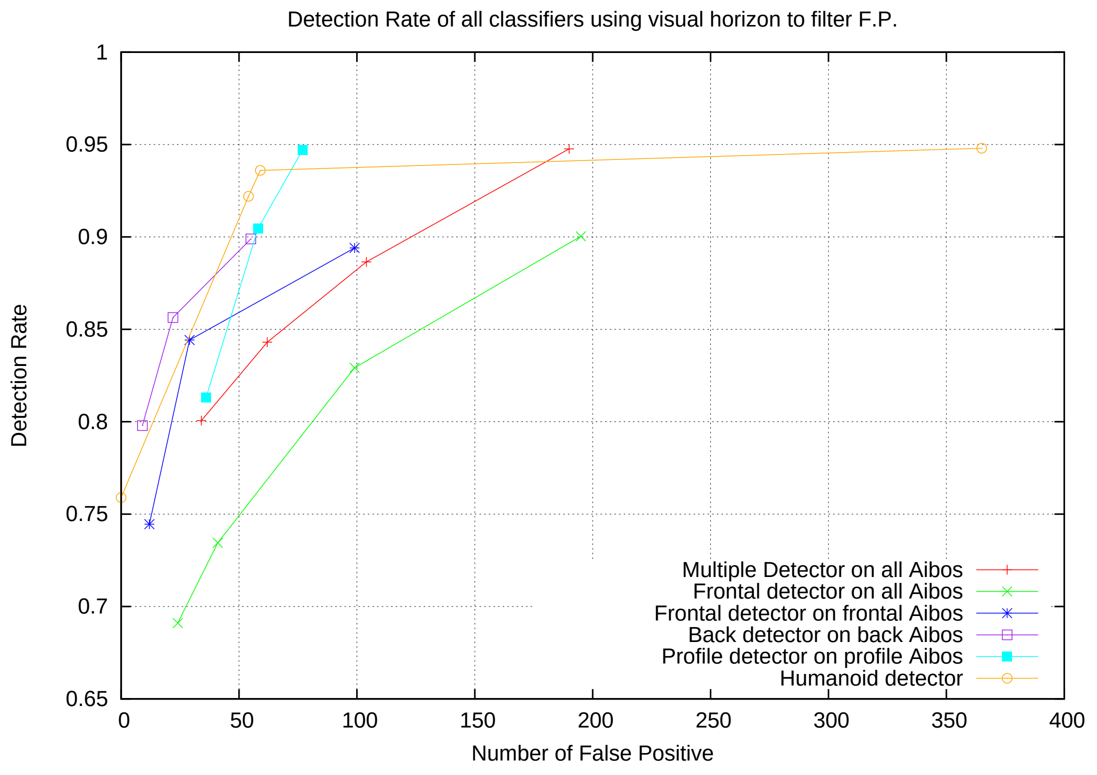


(b)

Figure 4.11: Some selected examples of humanoid robot detection with and without the use of context information (visual horizon, shown in green). In red are shown the false positives that were filtered using the visual horizon.



(a) not using visual horizon



(b) using visual horizon

Figure 4.12: ROC curves of the Aibo detector with (a) and without (b) the use of context information.

Hand Detection Results

The problem of hand detection is a complex one, as a hand is an object with a large number of degrees of freedom, i.e., a hand can present many different poses. Here we present results for detectors built to detect a few gestures (see Figure 4.13 for examples of the considered hand gestures). For each of these gestures, a cascade detector was built using the proposed learning framework. Figure 4.14 presents the ROC curves for the obtained hand gesture detectors, where it can be observed that the performance for some gestures (e.g. the gesture *five*) is clearly lower than for others (e.g. the gesture *fist*). Considering the complexity of the problem, the final gesture recognition and hand detection system also used tracking and skin detection methods (for details see [Franke et al., 2007]). Results of the complete system (including skin filtering, a tracking system, and a gesture classification module) are shown in Figure 4.15.

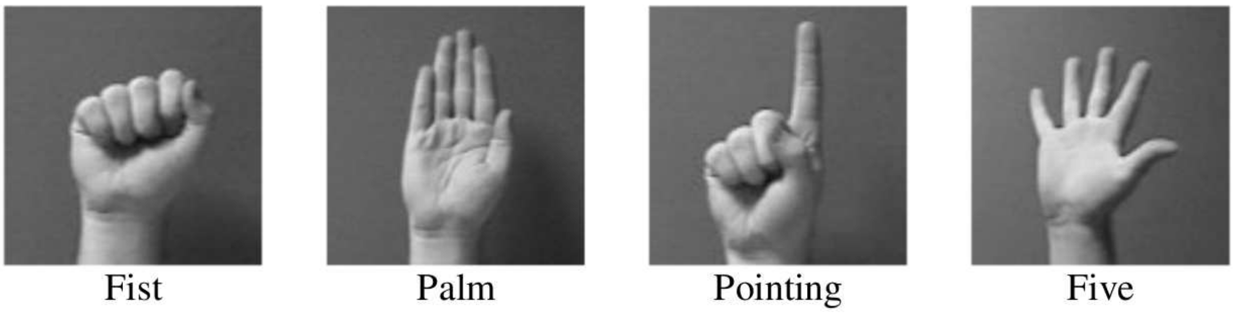


Figure 4.13: Examples of considered the hand gestures

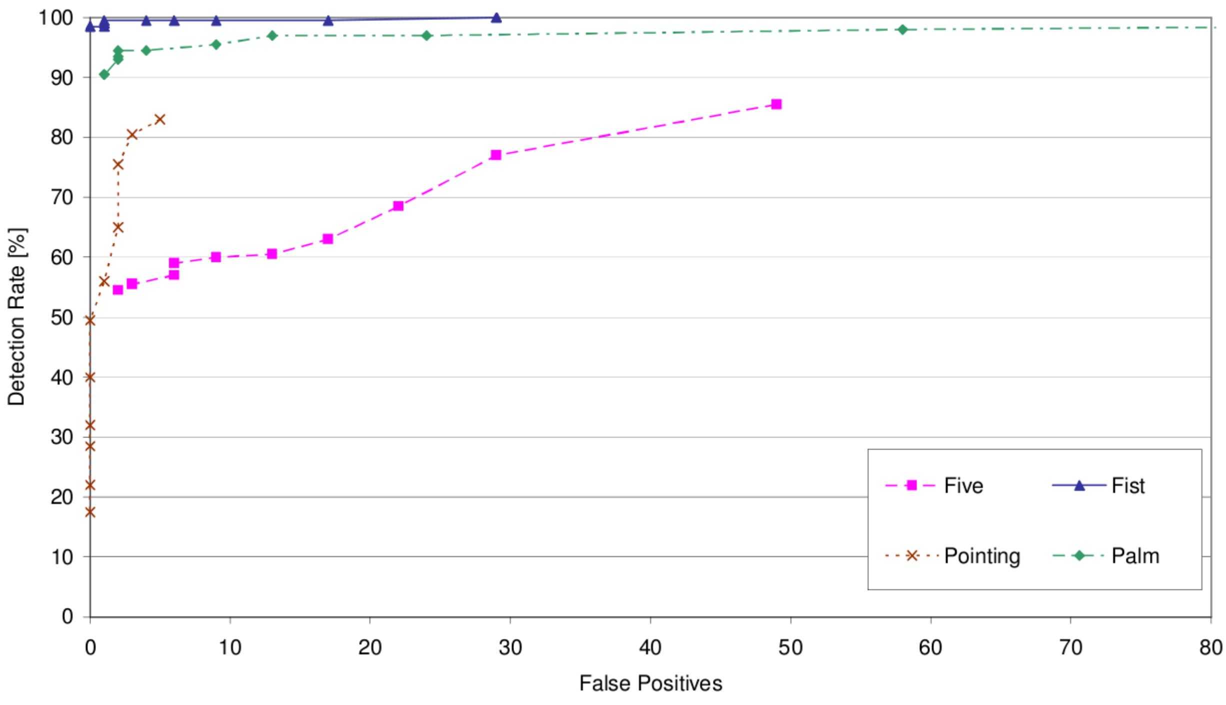


Figure 4.14: ROC curves of the hand gesture detector (does not include tracking).

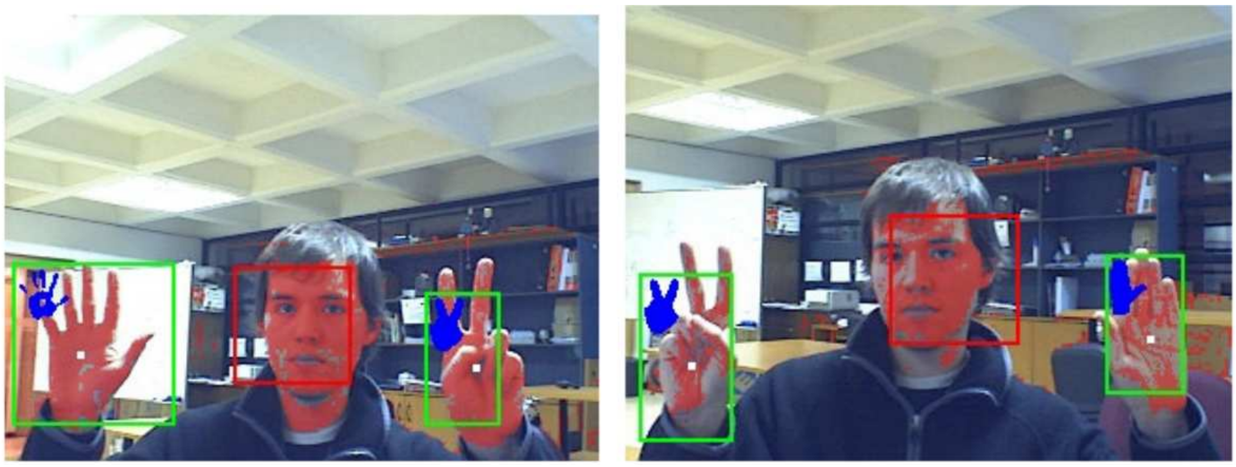


Figure 4.15: Example results of the hand gesture detection and recognition system. The five, victory, and palm gestures are detected (green boxes) and recognized (blue figures). Note the cluttered background, highlights, and skin-like colors.

4.5 Conclusion

An important goal of machine vision is to develop systems that can detect objects in cluttered backgrounds, with the ability of generalization across intra-class variability. In this context, we have proposed an efficient unified learning framework for object detection and classification using nested cascades of boosted classifiers. The most interesting aspect of this framework is the integration of powerful learning capabilities together with effective training procedures.

This framework is based on the use of Adaboost for the training of a nested cascade classifier, domain-partitioning for the design of weak classifiers, a procedure for handling the trade-off between the classification rates and the complexity of each layer, and rectangular and mLBP features. For the training of the detection systems, internal and external bootstrap is used, which together with feature sampling during training, combined use of rectangular features in the first two cascade layers, and mLBP features in the subsequent layers, and an adequate selection of the maximum FPR per layer allows us to train a face detection system in about 15 hours using in a Pentium 4 personal computer[e], which is considerably lower compared to previously presented work.

These framework has been used on the development a handful of object detection systems. First, face detection and eyes detection systems were built, and compared with similar state of the art systems. Our face detection system obtains the best-reported results on the BioID database, and is among the best reported results on the CMU-MIT database. Our eyes detection system obtains an important improvement over the best-reported results on the BioID database. For future comparisons with the here presented systems, a performance evaluation was carried out in images of the FERET database and in the new UCHFACE database. This new database includes the annotation of the faces, eyes (plus other landmarks), and gender, and it has been made available for the research community.

We have also used the proposed framework for building gender classification systems that were evaluated using the FERET, BioID, and UCHFACE databases. The main improvements over pre-

[e]Further implementation improvements used in the next chapter, including a faster implementation of the search of the thresholds *b*, the use of parallelism (OpenMP), and a faster (quad-core dual) processor computer, reduces the training time from 15 hours to less than 30 min.

vious works are: (1) the use of more suitable features for addressing this problem –mLBP features behave better than rectangular features–; (2) the usage of smaller face windows (24x24) which allows analyzing smaller faces, and (3) a faster processing, because, besides the eye alignment, we do not perform any geometric or photometric normalization. From the shown experiments it can be concluded that these systems achieve high accuracy in dynamical environments, and that they largely outperform SVM-based systems in terms of processing speed.

The obtained results in the case of eye detection show, not only a high detection rate, but also a high precision in the localization of the eyes with detection rates over 95% with less than 5% error in the position of the eyes (relative to the distance between the eyes).

In addition to the eye and face detection results, hand, profile cars, and robot detections results were presented. In the case of the profile cars, results on the UIUC database were presented, showing a performance close to the best results published so far. These results, together with the results of face and eye detection, show that the proposed methods can be applied to a wide range of problem.

As has been shown, the most interesting aspect of this framework is the possibility of building detection and classification systems that have a high processing speed and the proposed training methods have a very high training speed. The majority of obtained systems present high accuracy, and robustness, while the most difficult cases show good results, where improvements could be obtained using more appropriate features.

Chapter 5

Coarse-To-Fine Multiclass Classifiers[a]

Abstract

Building robust and fast object detection systems is an important goal of computer vision. A problem arises when several object types are to be detected, where the computational burden of running several object specific classifiers in parallel becomes a problem. Seeking to provide a solution to this problem, we extend the concept of nested boosted cascade classifiers to the multiclass case. We do this by introducing the so called *TCAS* classifier structure, which corresponds to a nested coarse-to-fine tree of multiclass nested boosted cascades that is designed to be used in multiclass and multiview object detection problems. For this we propose methods to build multiclass nested cascades of boosted classifiers, introduce the concept of coupled components in multiclass weak classifiers, and the use of coarse-to-fine (CTF) multiclass cascades. We also propose efficient training algorithms to build CTF Multiclass cascades and *TCAS*s. Results showing that the proposed systems scales well with the number of classes, both at training and running time are presented. The proposed algorithm is used to build a multiclass detector of hand fists and faces, and a multiview face detector. State of the art results are obtained on this last problem on the CMU rotated database. We show that the use of a *TCAS* can reduce considerably the processing time, and slightly increase the detection rate compared to the case when parallel classifiers are used. In addition, the training time of a multiclass *TCAS* takes only hours on a modern computer when few classes are used (e.g. $\sim$8 hours for 10 classes). These results show the robustness of the proposed classifiers and methods for building multiclass and multiview object detectors systems.

Keywords: Object detection, boosting, multiclass Adaboost, nested cascade classifiers, nested tree, *TCAS*, coarse-to-fine, face detection, coupled components, multiclass weak classifiers

[a]Sections 5.1 to 5.3 of the present chapter are based on the paper *[Verschae and Ruiz-del-Solar, 2008]*

Contents

5.1 Introduction

The development of robust and real-time object detection systems is an important goal of the computer-vision community. Some examples of applications that could make use of multiclass detection system include human computer interfaces (e.g. face and hands detection), autonomous drivings system (e.g. car, pedestrian and sign detection), and OCR in unconstrained environments (letters detection).

Other examples are the particular cases of multiview and multipose object detection, where each view-range (rotation with respect to the camera) and pose (deformation of the object) can be considered as a (sub)class. Face detection is a particularly interesting case for testing multiview detection systems, as the development of robust object and face detection systems is very important from the point of view of many applications, from robotics to multimedia retrieval and security applications. This problem is still unsolved and at the same time it is an interesting application to test object-detection systems.

In some cases the detection of multiple poses must be handle together with the problem of having different views. Examples where the presence of multiple poses is particularly difficult from the detection point-of-view are the problems of *a*) pedestrian detection, which can present many views (frontal, back, etc.) and many poses (standing, walking, etc.) and *b*) hand detection, which presents a large variety of poses and views (hands and arms have a large number of degrees of freedom).

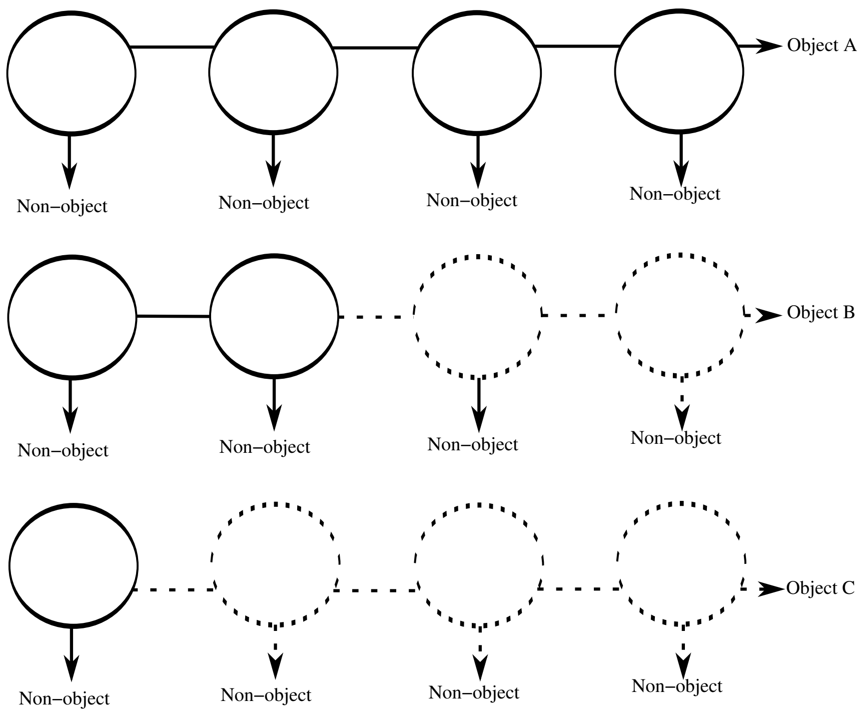


Figure 5.1: Multiples cascades. Example of 3 cascades, each with 4 layers. The output of each cascade represents a particular class. In the example the dashed lines and circles indicate layers of the cascades that are not evaluated during the evaluation of the cascades for a particular windows being classified.

For detecting all instances of an object class, the sliding window approach [Hjelms and Low, 2001] [Yang et al., 2002] (Figure 2.3.1, Chapter 2), requires to perform an exhaustive search over the window patches at different scales and positions of the image. The classification of all possible windows may require a high computational power. To achieve an efficient detection, less time should be spent on non-object windows than on object windows. This can be achieved using cascade classifiers [Viola and Jones, 2001]. In the case of a multiclass object detection problems, the most simple solution would be using several cascades working in parallel, with one cascade for each class (see Figure 5.1 for an example). Thus, a problem arises when several object types are to be detected, case where both, the processing time and the false positive rates, increase.

Most research groups have focused on building one-class object detection systems, mainly because it is still unclear what is the best way to extend binary classifiers to the multiclass case. In addition, multiclass systems must scale up with the number of classes in terms of computational efficiency, training time and robustness. Taking these requirements into consideration, we take the framework introduced in Chapter 4 and extend it to the multiclass case.

Contributions

Our work builds on the ideas of [Wu et al., 2004] and [Verschae et al., 2008b] (see Chapter 4) on one-class nested cascade classifiers, and on the ideas of [Huang et al., 2007] and [Torralba et al., 2007] on multiclass extensions of Adaboost. To deal with the multiclass object detection problem, we propose to build multiclass nested boosted trees and cascades. The proposed detector corresponds to a *multiclass* nested tree of multiclass nested cascades (we refer to it as *TCAS*).

One of its main novelties is that both, the tree and cascades have a *nested* structure, and they consist of multiclass classifiers. The training of the layers of the cascade is based on a multiclass version of Adaboost (called Vectorboost) proposed by [Huang et al., 2007], but here the nested structure of the cascades is also considered. The *TCAS* are designed using the multiclass cascades as building blocks. In addition we also introduce: *a*) the use of *coarse-to-fine* search in the target object space of the multiclass cascade, and *b*) the use of *coupled* components to train multiclass weak classifiers. The main differences with related work are:

- Instead of multiple one-class object detectors [Wu et al., 2004] [Jones and Viola, 2003b] [Schneiderman, 2004], the proposed system is based on multiclass cascade/tree classifiers, which allows to reduce considerably the processing time.
- Unlike [Torralba et al., 2007], the classifier has a coarse-to-fine structure (cascade and tree of cascades), and unlike [Huang et al., 2007] the classifier reuses information during the evaluation of the cascade/tree thanks to the use of a nested structure, allowing to obtain faster and more robust detectors.
- The training procedure scales well with the number of classes (unlike [Torralba et al., 2007] where the training time grows exponentially.)
- In contrast to [Wu et al., 2004] and [Huang et al., 2007] the search procedure is coarse-to-fine in the object target space, procedure which in our case has shown to allow faster and more robust detection results.

In addition, and contrasted to [Wu and Nevatia, 2007], our system is meant to build not only multiview detection systems, but also multiclass detection systems.[a] Also note that we focus on the problems of multiview face detection problem and multiclass object detection.

Chapter Structure

The structure of the present Chapter is as follows. The used multiclass version of Adaboost, including with three variants to train the multiclass weak classifiers, are described in Section 5.2. In Section 5.3 the proposed multiclass cascade structure and its training algorithms are explained. In Section 5.4 an improved multiclass nested cascade which uses a coarse-to-fine search on the object space is introduced. In Section 5.5 the proposed multiclass nested tree (*TCAS*) and the corresponding training and classification algorithms is presented. In Section 5.6 results of the proposed method are given, and in Section 5.7 conclusions are outlined.

5.2 Multiclass Adaboost and Multiclass Weak Classifiers

Following [Huang et al., 2007], the multiclass classifier used at each layer of the cascade has a vectorized form: [b]

$$\vec{H}(x) = \sum_{t=0}^{T} \vec{h}_t(\vec{f}_t(x)) \tag{5.1}$$

[a]We think that the work proposed here is complementary to [Wu and Nevatia, 2007], in the sense that the backwards partition done during the training by [Wu and Nevatia, 2007] could be also applied here.

[b]Notation: we use $\vec{H}(\cdot)$ to refer to a multivariate *function*, taking (a window patch) x as input, and with an output in $\mathbb{R}^M$. Note also that we will use $H(x,m)$ to refer to the component m of the output of $\vec{H}(x)$.

where in the simplest case, each component can represent a class or view. The training of each layer is performed using an algorithm similar to the one proposed in [Huang et al., 2007], however there are several differences that are presented in the following sections. The main differences are:

- A generalized version of Vectorboost [Huang et al., 2007] is presented (see Section 5.2.1), using objective regions that could be defined in a more general manner (not only intersections of half spaces as [Huang et al., 2007] do). Note that in the final implementation we use half spaces, and testing other options is left as future work.
- We note that the multiclass weak classifier to be used can have many different structures, and we identify two existing cases (independent components [Huang et al., 2007] and joint components [Torralba et al., 2007]), and propose a new one (coupled components, described in Section 5.2.3).

There are other differences compared to [Huang et al., 2007] which are not related to Vectorboost, but related to the training procedure and to the structure of the nested cascade. These differences are outlined in Sections 5.3 - 5.4.

5.2.1 A Multiclass Generalization of Adaboost

The basic idea behind this algorithm is to assign to each training example x_i an objective region in a vector space. The objective region is defined as the intersection of sub spaces (e.g. a set of half spaces), with each subspace defined by some parameters (e.g. a vector $\vec{a}$), and with a set of regions defined by a set $\mathbf{R}$ of parameters and a function $Q(\cdot,\cdot)$. Under this setting, a sample x, belonging to class Y_q and represented by a parameter set $\mathbf{R}_q$, is classified correctly if and only if:

$$\forall \vec{a} \in \mathbf{R}_q, Q\left(\vec{a}, \vec{H}(x)\right) \geq 0 \tag{5.2}$$

Therefore a class represented by $\mathbf{R}_q$ will be assigned to a new sample x, if this inequalities are fulfilled[c]. The proposed multiclass generalization of Adaboost is presented in Algorithm 5. The function $Q(\cdot,\cdot)$ can take different forms depending on the application and on previous knowledge about the classes. For example it could be a quadratic function, or the inner product as in [Huang et al., 2007], with $Q(\vec{a}, \vec{H}(x)) = \left\langle \vec{a}, \vec{H}(x) \right\rangle$.

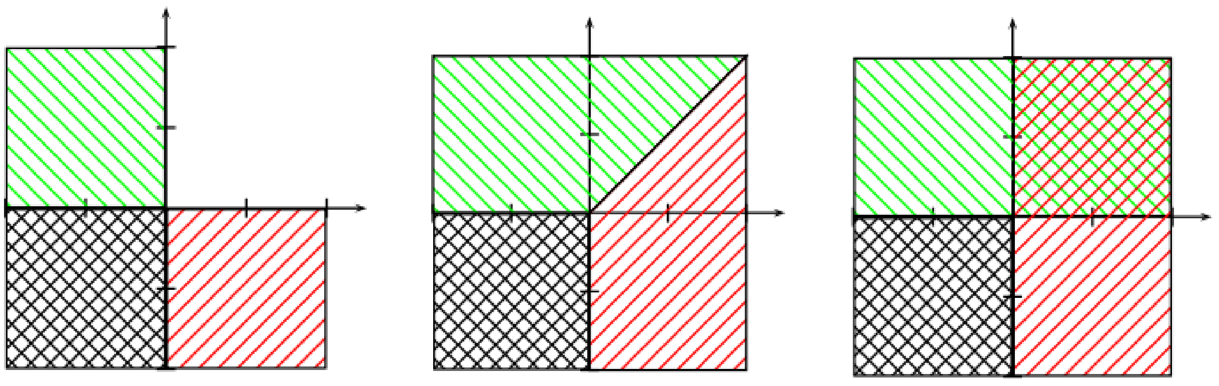
For simplicity, in the following we consider this last case $\left(Q(\vec{a}, \vec{H}(x)) = \left\langle \vec{a}, \vec{H}(x) \right\rangle\right)$. We also take M, the dimension of the vectors $\vec{a}$ as the number of classes we want to detect. Therefore $\vec{H} : \mathbb{R}^P \rightarrow \mathbb{R}^M$, with P the number of pixels on a image window. In this case, to determine if class Y_q can be assigned to an input x, it is necessary to test whether $\left\langle \vec{a}, \vec{H}(x) \right\rangle$ is positive for all vectors $\vec{a} \in \mathbf{R}_q$, i.e,. for all vectors associated to class Y_q.

This simplified setting is already very flexible as shown in the following examples. The Figure 5.2 and Table 5.1 present three possible setups for a *3-classes* classification problem in a two-dimensional target space. The most straight forward case is case (a), where the three classes have target spaces that do not overlap and where each target space is defined by two vectors. In this case the vectors encoding the target regions of each vector of each class are perpendicular to each

[c]In the one-class case a training example (x_i, y_i) with $y_i \in \{-1, 1\}$, is classified correctly if the margin is positive ($yH(x) \geq 0$), i.e. if $H(x)$ is in the correct half space. Therefore equation 5.2 can be interpreted as a multiclass extension of Adaboost, and the function $Q(\cdot,\cdot)$ as a multiclass extension of the concept of margin.

Algorithm 5 GENERALIZEDADABOOST(S,T)

```
Input: Training set $S = \{(x_i, \mathbf{V}_i)\}_{i=1,\dots,n}$, with $\mathbf{V}_i = \{\vec{V}_i^j\}_j$ a vector/parameter set representing the objective region associated to the example $x_i$.
Input: $T$ number of iterations
1: Initialize Weights: $w_{i,j}(0) = 1$
2: for $t = 0,\dots,T$ do
3:     Normalized weights $\{w_{i,j}(t)\}_{i,j}$ so that they add to one.
4:     Select $\vec{h}_t \in \mathbf{H}$ such that $Z_t$ is minimized:
           $$Z_t = \sum_{i=1}^{n} \sum_{j:V_i^j \in \mathbf{V}_i} w_{i,j}(t) \exp\left(-Q\left(\vec{V}_i^j, \vec{h}_t(x_i)\right)\right)$$
5:     Update weights:
           $$w_{i,j}(t+1) = w_{i,j}(t) \exp\left(-Q\left(\vec{V}_i^j, \vec{h}_t(x_i)\right)\right)$$
6: end for
Output: return $\vec{H}(x) = \sum_t \vec{h}_t(x)$
```



(a) Disjoint and non overlapping (b) Joint boundary (c) Ovelapping

Figure 5.2: Examples of possible target spaces for Vectorboost with 2 objects classes and a non-object class (see also Table 5.1)

other, and define a quadrant of the two-dimensional space. Case (b) corresponds to three classes with target spaces that do not overlap, but in this case each pair of target spaces have a common boundary. Case (c) is an example of the setting we use in the present work (here is shown for a 2-classes detection problem), and corresponds to the case where the vectors $\vec{a} \in \{\mathbf{e} \cup -\mathbf{e}\}$, with $\mathbf{e}$ the canonical base.

In this last case, two "object" classes share a part of the target space, but do not overlap with the third "non-object" class. This means that the "non-object" class is a common *negative* class to the two object classes, and that the overlapping of the target space of two of the classes relaxes the classification problem. This relaxation allows to focus on what we are really interested in, which is on detecting the "object" classes, i.e. focusing on discriminating the two object classes from "non-objects" , and not on discriminating between the three classes as done in a "common" multiclass

Table 5.1: Examples of possible target spaces for Vectorboost with 2 objects classes and a non-object class (see also Figure 5.2)

(a) Disjoint and non overlapping

Class	Vector Set	Objective region	Region on figure
Object 1	$(+1,0);(0,-1)$	$x>0, y<0$	Red
Object 2	$(-1,0);(0,+1)$	$x<0, y>0$	Green
Non Object	$(-1,0);(0,-1)$	$x<0, y<0$	Dashed Grey

(b) Joint boundary

Class	Vector Set	Objective region	Region on figure
Object 1	$(+1,0);(+1,-1)$	$x>0, x>y$	Red
Object 2	$(-1,0);(-1,+1)$	$y>0, y>x$	Green
Non Object	$(-1,0);(0,-1)$	$x<0, y<0$	Dashed Grey

(c) Overlapping

Class	Vector Set	Objective region	Region on figure
Object 1	$(1,0)$	$x>0$	Red
Object 2	$(0,1)$	$y>0$	Green
Non Object	$(-1,0);(0,-1)$	$x<0, y<0$	Dashed Grey

setting. Note that even if we are allowing the object classes to “overlap”, we are not completely mixing them, which at the end allows to discriminate among them with a good accuracy.[d]

Before continuing, and as explained in [Huang et al., 2007], it is important to mention that in Algorithm 5 the training set $S=\{(x_i,\mathbf{V}_i)\}_{i=1,\ldots,n}$ can be replaced by a training set of the form $\hat{S}=\{(x_i,\vec{a}_i)\}_{i=1,\ldots,\hat{n}}$. In $\hat{S}$ each example x_i from S appears $|\mathbf{V}_i|$ times, each time with a different vector $\vec{a}_i \in \mathbf{V}_i$. This can be done thanks to the form of the minimization of Z_t (see line 4), where the terms can be re-arranged in only one summation. Therefore, to simplify the notation, in the following the algorithms are presented assigning only one vector to each training example, but the general case still holds.

5.2.2 Feature Sharing

As done by [Torralba et al., 2007] [Huang et al., 2007], to speed up the classification process and to simplify the learning process, we assign a single feature to each weak classifier:

$$\vec{h}_t(\vec{f}_t(x)) = \vec{h}_t(f_t(x)) \ ,$$

i.e., all components share feature evaluations. This is a key point to gain on processing time. For example, in the setting used by [Torralba et al., 2007], the sharing of features and weak classifiers allows to considerably reduce the processing time when using a multiclass classifiers, where they show that the processing time grows logarithmically with the number of classes.

In the following, we will use $h_t(f_t(x),m)$ to refer to the component m of a weak classifier $\vec{h}_t(f_t(x))$ to simplify the notation.

[d]This framework can be use in settings where each training example has is own target region. Thus, previous knowledge can be encoded, e.g. representing its class label, and encoding its pose and view with respect to the camera.

5.2.3 Multiclass Weak Classifiers

The multiclass weak classifiers can have many different structures and this does not need to be related to the boosting classifier being used. This means that the boosting algorithm and the multiclass weak classifier structure do not need to be connected (as done in [Torralba et al., 2007] [Huang et al., 2007]) and that other boosting approaches, such as Gentleboost [Lienhart et al., 2003], Logiboost [Friedman et al., 1998], could be used with the multiclass weak classifiers instead of the proposed generalized version of Adaboost.

Taking this into consideration, we identify three different ways of selecting the functions $h_t(f_t(x),m)$. In the first one, that we call *independent* components, the components of $\vec{h}_t(f_t(x))$, $\{h_t(f_t(x),m)\}_{m=1,\dots,M}$, are chosen independently of each other (as in [Huang et al., 2007]).

In the second case, *joint* components, the components are chosen jointly (like in [Torralba et al., 2007]): the same function $\bar{h}_t(\cdot)$ is used for different components/classes and the remaining components output a zero value:

$$h_t(f_t(x),m) = \beta_t^m \bar{h}_t(f_t(x)), \beta_t^m \in \{0,1\}. \tag{5.3}$$

In the third case, we introduce the concept of *coupled* components, where components share a common function, but for each component this function is multiplied by a different scalar value ${\gamma_t}^m$:

$$h_t(f_t(x),m) = {\gamma_t}^m \bar{h}_t(f_t(x)), {\gamma_t}^m \in \mathbb{R}. \tag{5.4}$$

This last case, *coupled* components (Equation 5.4), resembles the case of joint components (Equation 5.3), but it presents several advantages: its training time is much faster (unlike [Torralba et al., 2007] it does not need a heuristic search), it is scalable with the number of classes, and as we will see later, it is much more accurate. In some cases *coupled* components also presents advantages over independent components, e.g. less parameters need to be estimated, which can help to avoid over-fitting, in particular when small training sets are used (this is shown in [Torralba et al., 2007] for joint components). The use of fewer parameters can be also useful in the cases when the memory footprint is important, for example, when the detector is to be used in mobile devices or robots with low memory capacity. Note, however, that there is a trade-off, not only between the complexity of the weak components and the size of the training set, but also the size of the boosted classifier.

Optimization Problem

As in Chapter 4, the weak classifiers are designed after the *domain-partitioning weak hypotheses* paradigm [Schapire and Singer, 1999]. Each feature domain $\mathbb{F}$ is partitioned into disjoint blocks $F_1,\dots,F_J$, and a weak classifier $h(f(x),m)$ will have an output for each partition block of its associated feature. During training, the use of coupled, independent or joint components, together with domain partitioning classifiers leads to different optimization problems that are outlined in the following.

As part of the optimization problem, in all three cases, the values $W_l^{j,m}$ with $j = \{1,\dots,J\}$, $m = \{1,\dots M\}$, and $l = \{-1,+1\}$ need to be evaluated, where $W_l^{j,m}$ represents the bin j of a weighted histogram (of the feature values) for the component m for "positive" ($l = 1$) or "negative" ($l = -1$) examples:

$$W_l^{j,m} = \sum_{i,j:\, f(x_i)\in F_j \wedge (\vec{a}_i)_m = l} w_{i,j}, \tag{5.5}$$

with $(\vec{a}_i)_m$ representing the value of the component m of the vector $\vec{a}_i$.

It is important to note that the evaluation of $\{W_l^{j,m}\}_{m,j,l}$ takes order O(N), with N the number of training examples, and does not depend on J (number of partitions) nor on M (number of classes/components). This linear dependency, on N only, is very important to keep a low computational complexity of the training algorithm.

In contrast to the one-class case, the output of the domain partitioning weak classifier also depends on the component of the multiclass classifier, therefore the output of the classifier, for a feature f, is: $h(f(x),m) = c_{j,m} \in \mathbb{R}$ such that $f(x) \in F_j$.

For each weak classifier, the value associated to each partition block ($c_{j,m}$), i.e. its output, is selected to minimize Z_t:

$$\min_{c_{j,m}\in\mathbb{R}} Z_t = \min_{c_{j,m}\in\mathbb{R}} \sum_{j,m} W_{+1}^{j,m} e^{-c_{j,m}} + W_{-1}^{j,m} e^{c_{j,m}} \tag{5.6}$$

and the value of $c_{j,m}$ depends on the kind of weak classifier being used.

In the case of independent components, this minimization problem has an analytic solution (using standard calculus):

$$c_{j,m} = \frac{1}{2}\ln\left(\frac{W_{+1}^{j,m} + \varepsilon_m}{W_{-1}^{j,m} + \varepsilon_m}\right) \tag{5.7}$$

with ε_m a regularization parameter. Note that the output components are independent, i.e., the values of c_{j,m_1} and c_{j,m_2}, for $m_1 \neq m_2$, are calculated independently and do not depend on each other.

It is important to note how this regularization value, ε_m, is selected. In [Schapire and Singer, 1999], in a two class classification problem, is shown that it is appropriate to choose $\varepsilon \ll \frac{1}{2J}$, with J the number of partitions. The authors recommended to use ε on the order of $1/n$, with n the number of training examples. If the same analysis is done in the multiclass setting used here, the value of ε_m should be of the order of $1/n_m$, with n_m the number of training examples for class m (this analysis is straight forward from Equation (11) on [Schapire and Singer, 1999]). This corresponds to smoothing the weighted histograms taking into account the number of training samples used to evaluate it.

In the case of weak classifiers with joint components, the optimization problem also has an analytical solution, but it requires to solve a combinatorial optimization problem. More formally, in this case $c_{j,m} = \beta_m c_j$, with $\beta_m \in \{0,1\}$. The optimization problem to be solved is:

$$\min_{c_j\in\mathbb{R},\beta_m\in\{0,1\}} Z_t = \min_{\beta_m\in\{0,1\}} \min_{c_j\in\mathbb{R}} \sum_{j,m} W_{+1}^{j,m} e^{-c_j\beta_m} + W_{-1}^{j,m} e^{c_j\beta_m} \tag{5.8}$$

In order to obtain the optimal value of this problem, 2^M problems of the form

$$\min_{\hat{c}_j\in\mathbb{R}} \sum_j \hat{W}_{+1}^j e^{-\hat{c}_j} + \hat{W}_{-1}^j e^{\hat{c}_j} \tag{5.9}$$

must be solved, with each of these problems having a solution of the form

$$\hat{c}_j = \frac{1}{2}\ln\left(\frac{\hat{W}_{+1}^j + \varepsilon}{\hat{W}_{-1}^j + \varepsilon}\right), \text{with } \hat{W}_l^j = \sum_{m:\beta_m=1} W_l^{j,m}, l = \{-1,+1\} \tag{5.10}$$

In the case of coupled components the optimization does not have an analytical solution, and the optimization problem to be solved is:

$$\min_{c_j,\gamma_m \in \mathbb{R}} Z_t = \min_{c_j,\gamma_m \in \mathbb{R}} \sum_{j,m} W_{+1}^{j,m} e^{-c_j \gamma_m} + W_{-1}^{j,m} e^{c_j \gamma_m} \tag{5.11}$$

It is important to note that this is a convex optimization problem, because it is a summation of weighted convex functions, where the weights are positive (weighted sum of exponential functions). In practice we use a quasi-Newton optimization method called L-BFGS [Liu and Nocedal, 1989], which is a BFGS variant designed for large scale problems with limited memory. Each iteration of L-BFGS is quadratic on the number of variables and does not explicitly stores or evaluates the Hessian matrix, which could be expensive when the number of variables is large (in our case the number of variables is $M + J$ variables, with, for example M = 5 and $J = 511$). Note that the gradient of Z_t can be evaluated efficiently without any approximation, which helps to have a faster convergence.

In the three variants (coupled, joint and independent), the outputs of a weak classifier ($c_{j,m}$), obtained during training, are stored in look-up-tables (LUT) for speeding up the evaluation. Thus, a weak classifier will be defined by $h(f(x),m) = h_{\mathrm{LUT}}[x,m] = \mathrm{LUT}[index(f(x)),m]$, with $index(\cdot)$ a function that returns the index (block) associated to the feature value $f(x)$ in the LUT for class m. After having evaluated the feature value, when using equally sized partitions and independently of the number of blocks, this implementation allows to compute the weak classifiers in constant time (the same time that takes to evaluate decision stumps).

Table 5.2[e] presents a summary of the optimization problems that have to be solved in each case (given that Domain Partitioning classifiers are used), as well as the number of parameters that has to be estimated, and the order (big O) of the training algorithm. Recall that the this optimization has to be solved at each iteration of Adaboost for each possible feature being considered (line 5, Algorithm 5), so this step is very important to have an efficient training algorithm. Note also that the final training time depends on how many weak classifiers are selected, which depends on how fast the boosting training algorithm converges for each case.

From Table 5.2 can be observed that in all variants the dependency on the number of training examples is linear, which is very important, because in many cases the number of training examples will grow when more classes are being considered ($N = \sum_{m=1}^{M} n_m$, with n_m the number of training examples per class). In terms of the dependency on the number of classes, in the case of *joint* components the training time for one weak classifier grows exponentially with the number of classes, while for *coupled* components grows quadratically and for *independent* components grows linearly. Another thing that is presented in Table 5.2 is the number of variables that must be estimated and stored. The case that requires to estimate the largest number of variables is independent components, and it may lead to over-fitting if small training sets are used; this also may be important if the required memory becomes too large.

5.3 Multiclass Nested Cascades

As previously mentioned we propose to use a multiclass cascade with a *nested* structure. Figure 5.3 presents a diagram of an example of a multiclass cascade. Note that this diagram does not show the nested structure of the multiclass cascade.

[e]Real Adaboost is designed for two-class classification problems and it is shown for reference.

Table 5.2: Comparison of weak classifiers training methods. Domain partitioning classifiers are used. J: number of partitions (bins); M: number of classes; N: number of training examples. See text for comments.
* To be used for 1-class object detection problems only.

Training method	Variables $j=1,\dots,J$, $m=1,\dots,M$	Number of variables	Solution	Order $O()$
Real Adaboost*	$c_j \in \mathbb{R}$	J	Analytic	$J+N$
Independent	$c_{j,m} \in \mathbb{R}$	JM	Analytic	$JM+N$
Joint	$c_j \in \mathbb{R}$, $\beta_m \in \{0,1\}$	J+M	Analytic	$J2^M+N$
Coupled	$c_j \in \mathbb{R}$, $\gamma_m \in \mathbb{R}$	J+M	Gradient Descent	$(J+M)^2+N$

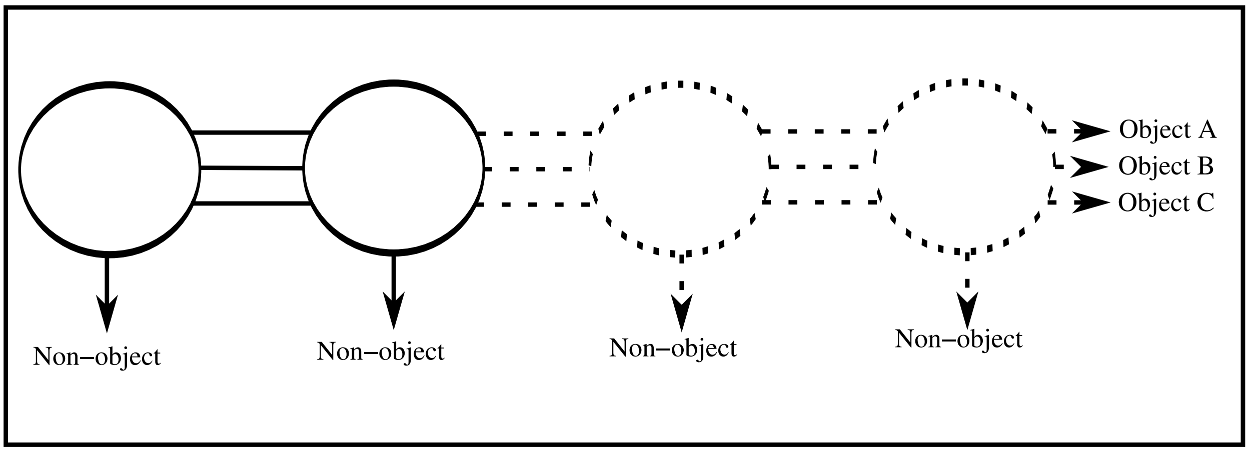


Figure 5.3: Multiclass cascade. Example of a multiclass cascade consisting of 4 layers, with boosted classifier of 3 components, and where the output encodes 3 classes. In the example the dashed circles indicates layers not being evaluated by the cascade for a particular windows being classified.

As in Chapter 4, the nested structure of the cascade is achieved by reusing information of a layer in the following ones. This is done by letting the classifier at layer k of the nested cascades be:

$$\vec{H}_k(x) = \vec{H}_{k-1}(x) + \sum_{t=0}^{T} \vec{h}_{t,k}(f_{t,k}(x)) - \vec{b}_k \tag{5.12}$$

$$\text{with } \vec{H}_0(x) = 0.$$

The training of a layer of a multiclass nested cascades is presented in Algorithm 8. The main advantages of using nested cascades are that the obtained cascades are more compact and more robust than in the non-nested case (see Chapter 4, and [Huang et al., 2005]).

5.3.1 Multiclass Nested Cascade Training

The training of the complete cascade system is presented in the TRAINCASCADENESTEDMC algorithm (see Algorithm 7). It consists of iteratively collecting “negative” training examples and then training and adding the layers to the cascade. To collect the "negative" examples, we propose to use a multiclass extension of the BOOTSTRAPP algorithm (see Chapter 4, Algorithm 2) that is presented in algorithm BOOTSTRAPPMC (see Algorithm 9). To the best of our knowledge, *multiclass* nested cascades and *multiclass* bootstrapping procedures have not been previously used.

The algorithms TRAINCASCADENESTEDMC and TRAINLAYERNESTEDMC are basically the same as in the one-class case (see Chapter 4), but modifications are needed in order to take into account the multiclass setting. The main differences appear in the minimization done at the training of the layers of the cascade (TRAINLAYERNESTEDMC), during the bootstrapping (BOOTSTRAPPMC) and during the validation of the boosted classifier (VALIDATECLASSIFIERMC). The minimization done at TRAINLAYERNESTEDMC was already presented at the previous section, and depends on the kind of multiclass weak classifier being used (independent, coupled or joint components). The remaining relevant differences are present in the following.

Boostrapping

The training of a nested cascade requires collecting a new "negative" training set (line 5, Algorithm 7) prior to the training of a new layer, a procedure that must be done considering the multiclass structure of the classifier. The multiclass BOOTSTRAPPMC algorithm that we use is presented in Algorithm 9. The core of the algorithm corresponds to lines 4-6. In line 4, each half space (represented by $\vec{a}_j$) associated to any "positive" class is selected. Afterwards, in line 5, it is verified if the current sample x^* was classified in the wrong half space, and if so, the example is added to the set of negative examples S (line 6). Note that the target region assigned to each negative examples (line 6) is defined only by the complement of the wrongly classified regions, and not by the "negative" region (for all classes), which is a relaxation of equation 5.2. This assigns, to each collected "negative" example, a region that corresponds to the intersection of the complement of the half spaces of those positive classes (components) for which the previously trained classifier gave a wrong answer (see lines 4-6 of Algorithm 9). This corresponds to train each layer of the cascade, to discriminate from "negatives" only considering the errors made by previous layers. Note that, as done in the one-class case (Chapter 4), internal and external bootstrapping can also be used, but here this is omitted for the sake of clarity.

Processing Speed and Accuracy of a Layer of the Multiclass Cascade

As in the one-class case (Chapter 4) we propose a criterion and a procedure to handle the trade-off between the accuracy of the classifier (false positive rate and detection rates) and the processing speed of the classifier. The algorithm VALIDATECLASSIFIERMC (Algorithm 10) presents the proposed solution, which basically defines a criterion for determining when to stop adding weak classifiers. This is done by searching for the optimal value of $\vec{b}_k$ and validating that the classifier fulfills the required minimum true positive rate (*tpr*) and maximum false positive rate (*fpr*) considering the multiclass setting. Compared to the one-class case there are two important differences. First, it requires to find $\vec{b}_k$, which we solve by simply using a vector of the form $b_k\vec{1}$ in order to reduce the search space (note that a more complex method could be used). This allows to have an efficient implementation, which corresponds to an exponential search followed by a binary search. This search is performed after each iteration of the boosting processes, so it should not be too time consuming.

Second, and more importantly, the criteria to be minimized in the mutilclass case involves having a maximum *fpr* and a minimum *tpr* per class, which means that multi-objective optimization problem being solved now has more objectives: a *fpr* per class, a *tpr* per class and the number of weak classifiers being added. In the one-class case this was solved by searching, at each iteration of Adaboost, a b_k that fulfilled requirements on the *fpr* and *tpr* rates for that layer.

At each layer we are mainly interested on reducing the total (average) number of false positives, and not the false positives associated to a particular class. Therefore, in the multiclass setting, we consider a restriction on the *fpr* per layer, and not per class, as one sample may be a false positive for more than one class.On the contrary, it is important that the *tpr* per class are balanced, and in particular that no class has a too low true positive rate. To solve this we have defined a criterion based on the product of the *tpr* of each class. Remember that the *tpr* has values in $[0,1]$ (the closer to one the better). Using the product allows to have a *tpr* that is not too low for any class and more importantly the product penalizes more strongly lower *tpr*.[f] This criteria can be be seen as putting a threshold on the average of $log(tpr)$, which clearly penalizes low true positives rates more strongly than high true positive rates.

Stop Condition

In addition we have added a stop condition to TRAINLAYERNESTEDMC (see Algorithm 8, line 4), which is needed when the learning algorithm is not converging fast enough or not converging at all. The stop condition can take several forms. We simply verify that t is not to large (i.e. $|t < T_{max}|$), but it could be a condition on the evolution of Z_t, for example it could be a condition on $|Z_t - Z_{t-1}| \leq \theta_t$ or $Z_t \leq \theta_t$ as done by [Wu and Nevatia, 2007]. This condition becomes important later (Section 5.5), when is used to trigger the splitting of a node during the training of a *TCAS*.

Algorithm 6 Multiclass nested cascade - Auxiliary Functions and Notation

```
1: Evaluation Rates
   $tpr(H,P)$: true positive rate of classifier $H$ evaluated on the set $P$.
   $fpr(H,N)$: false positive rate of classifier $H$ evaluated on the set $N$.
2: $(\vec{V})_m$ refers to the component $m$ of the vector $\vec{V}$.
3: $\vec{a} \odot \vec{b}$: point-wise product between $\vec{a}$ and $\vec{b}$, also know as the Hadamard product:
   $$\left(\vec{a} \odot \vec{b}\right)_m = (\vec{a})_m (\vec{b})_m$$
4: $H(x,m)$ refers to the component $m$ of the output of the function $\vec{H}(x)$.
```

5.3.2 Analysis & Results

To evaluate the proposed learning algorithms, we took the problem of multiview face detection and considered different views as different classes by taking frontal faces and rotating them on multiples of 45 degrees (in-plane rotation) to define new classes. Table 5.3 shows a summary of the 8 defined classes.

In order to evaluate how well the system scales with the number of classes, we considered 4 problems, each one having an increasing number of classes (1, 2, 4 and 8 classes). In all cases the

[f]We did some tests using criterion based on the minimum *tpr* among all classes and the average *tpr* of all classes, but the first case was too strict, and the second generated solutions with very high *tpr* for some classes and very low *tpr* for “difficult” classes.

Algorithm 7 TrainCascadeMC$(H_{init}, S, B, F, F_C, D, PV, NV, NB, T_{max})$

```
Input: Usually $\vec{H}_{init} = \vec{0}$
Input: Set of positive examples $S = \{(x_i, \vec{a}_i)\}_{i=1,\ldots,n}$
Input: Bootstrapp set $B$
Input: Maximum false positive rate per node $F$
Input: Target false positive rate $F_C$
Input: Minimum detection rate per node $D$
Input: Positive validation set $PV$
Input: Negative validation set $NV$
Input: Number of bootstrapped examples $NB$
Input: Maximum number of iterations $T_{max}$
 1: $k = 1$, $F_k = 1$
 2: $S_{B_0} = B$
 3: $\mathbf{R} = \cup_{i=1}^{n} \{\vec{a}_i\}$
 4: while $F_k > F_C$ do
 5:     $S_{B_k} = \text{BootstrappMC}(S_{B_{k-1}}, \vec{H}, \mathbf{R})$
 6:     $\vec{H}_k = \text{TrainLayerNestedMC}(S \cup S_{B_k}, D, F, \vec{H}, PV, NV, T_{max})$
 7:     $\vec{H} \leftarrow \vec{H} \cup \vec{H}_k$
 8:     $F_k \leftarrow$ Evaluate current cascade $\vec{H}$
 9:     $k \leftarrow k + 1$
10: end while
Output: Trained Cascade $\vec{H}$
```

Algorithm 8 TrainLayerNestedMC$(S, F, D, \vec{H}_{init}, PV, NV, T_{max})$

```
Input: Training Set $S = \{(x_i, \vec{a}_i)\}_{i=1,\ldots,n}$
 1: $t \leftarrow 0$
 2: $\vec{H}_0(\cdot) \leftarrow \vec{H}_{init}(\cdot)$
 3: $w_0(i) \leftarrow \exp(-\vec{a}_i \cdot \vec{H}_0(x_i))$ , $i = 1, \ldots, n$
 4: while $\left( \left( !\text{ValidateClassifierMC}(\vec{H}_t, k, PV, NV, F, D) \right) \wedge (!StopCondition(t, T_{max})) \right)$ do
 5:     Normalize $\{w_t(i)\}_{i=1,\ldots,n}$ s.t they add to one
 6:     Select $\vec{h}_t \in \mathbf{H}$, $f_t \in \mathbf{F}$ that minimizes
            $$Z_t = \sum_i w_t(i) \exp\left( - \left\langle \vec{a}_i, \vec{h}_t(f_t(x_i)) \right\rangle \right)$$
 7:     Update weights:
            $$w_{t+1}(i) = w_t(i) \exp\left( - \left\langle \vec{a}_i, \vec{h}_t(f_t(x_i)) \right\rangle \right)$$
 8:     $t \leftarrow t + 1$
 9: end while
Output: Trained layer $\vec{H}(\cdot) = \sum_{t=1}^{T} \vec{h}_t(\cdot) - \vec{b}_k$
```

Algorithm 9 BOOTSTRAPPMC($S_B, H, NB, \mathbf{R}$)

```
Input: Bootstrapp set $S_B$
Input: Number of negative examples $NB$
Input: Classifier $H$
Input: Vector set $\mathbf{R}$
 1: $S = \{\emptyset\}$, $i = 0$
 2: while $i < NB$ do
 3:     $x^* \leftarrow$ sample $S_B$
 4:     for each $\vec{a}_j \in \mathbf{R}$ do
 5:         if $\left\langle \vec{a}_j, \vec{H}(x) \right\rangle > 0$ then
 6:             $S \leftarrow S \cup \left\{ (x^*, -\vec{a}_j) \right\}$
 7:             $i \leftarrow i + 1$
 8:         end if
 9:     end for
10: end while
Output: $S$
```

Algorithm 10 VALIDATECLASSIFIERMC($\vec{H}_C^k, k, \{PV_m\}_{m=1,\ldots,M}, NV, fpr_{Max}, tpr_{Min}$)

```
Input: $H_C^k$: Classifier
Input: $k$: current layer index
Input: $M$: Number of classes
Input: $PV_m = \{(x_i, \vec{a}_i)\}_{i=1,\ldots,n_m}$: Validation set of class $m$
Input: $NV$: Negative validation set
Input: $fpr_{Max}$: Maximum false positive rate per layer
Input: $tpr_{Min}$: Minimum true positive rate per layer (per class)
 1: for $b_l \in B = \{b_1, \ldots, b_L\}$ * do
 2:     $\vec{H} \leftarrow \vec{H}_C^k - b_l \vec{1}$
 3:
        $$p \leftarrow \left( \left( fpr(\vec{H}, NV) \leq {fpr_{Max}}^k \right) \wedge \left( \prod_{m=1,\ldots,M} tpr(\vec{H}, PV_m) \geq {tpr_{Min}}^{kM} \right) \right)$$
 4:     if $p$ then
 5:         $\vec{b}_k \leftarrow b_l \vec{1}$
 6:         $\vec{H}_C^k \leftarrow \vec{H}_c^k - \vec{b}_k$
 7:         return TRUE
 8:     end if
 9: end for
10: return FALSE
Output: $\vec{H}_C^k$
    * This search is implemented using an exponential search followed by a binary search
```

Table 5.3: Problem setup and training parameters. TPR(FPR): true(false) positive rate per class. ETPR: Expected TPR for a cascade of 14 layers; under this configuration a FPR of $4.13e^{-6}$ is expected for a complete cascade.

Problem	1 class	2 classes	4 classes	8 classes
Classes [degrees]	$0°$	$0°, 180°$	$0°, 180°, 90°, 270°$	$0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°$
TPR per layer per class	0.999	0.999	0.999	0.999
Min TPR product per layer	0.999	0.998	0.996	0.992
Max FPR per layer	0.35	0.35	0.35	0.35
ETPR	0.987	0.987	0.987	0.987

number of training examples per class (face views) was 5000. The number of training examples for the negative class, selected during bootstrapping, was 5000M, with M the number of object classes. In this way we got an equal number of “positive” and “negative” examples.

To speed up the training, we used feature sampling and we consider rectangular features in the first two layers, and mLBP in the subsequent layers, as done in Chapter 4. For the training of the multiclass weak classifiers, we considered independent, joint and coupled weak classifier components.

The obtained results are presented as ROC curves (Figure 5.4), Detection Rates (DR), number of False Positives (FP) and processing times (Table 5.4). The evaluation was done using the BioID database, which consists of 1521 images (of 384x286 pixels), each one containing one frontal face; the images were rotated to the eight mentioned rotations.

Figure 5.4 presents the obtained detection results for class 1 (frontal faces) using the multiclass classifiers, omitting the output of other classes (i.e. false positives given for other classes are not considered). Results for other classes are very similar. In the one-class case (Figure 5.4, top-left) the three weak learning algorithms gave very similar results, with coupled components working slightly better. In the cases of the two-classes (Figure 5.4, top-right), and four-classes (Figure 5.4, bottom-left) classifiers, best results are obtained by coupled components, followed by independent and joint components. In the case of the 8-class classifier (Figure 5.4, bottom-left), best results are obtained by independent components, followed by the coupled ones. Table 5.4 presents a summary of these results for 5 FP.

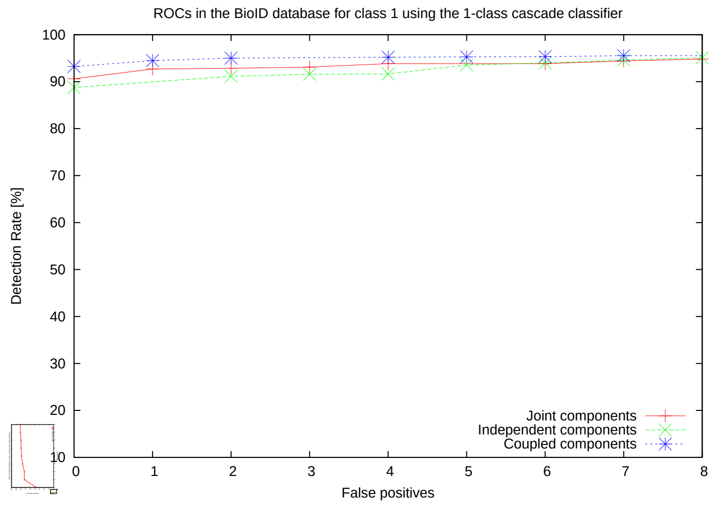
An interesting result is that for coupled components, the four-classes and two-classes classifier worked better than the independent one and the joint one, even when the classifier was trained to detect half of the classes (see Table 5.4). However, for the eight-classes case, best results are obtained by independent components, although they are not very good (60% DR for 5 FP).

Table 5.4 also summarises some results regarding the processing time and the size of the boosted classifiers. In terms of the training time, all methods took about two hours for the one-class case. In the eight-classes case it took 51, 56 and 109 hours for the coupled, independent and the joint components respectively (in a 3 GHz Pentium 4 computer). In the 8-classes case, the training time of coupled components and independent components is twice as fast as for joint components; this result reflects the analysis of Table 5.2 on the complexity of learning for the different multiclass weak classifiers. Regarding the processing time, jointly trained components is the fastest one, growing very slowly with the number of classes. In the case of coupled components, the processing time grows almost linearly with the number of classes. In the case of independent components the processing speed remains very fast for 2 and 4 classes (about 0.4 sec.), but it goes up to 4.4 seconds (12 times slower than for the 4 class case).

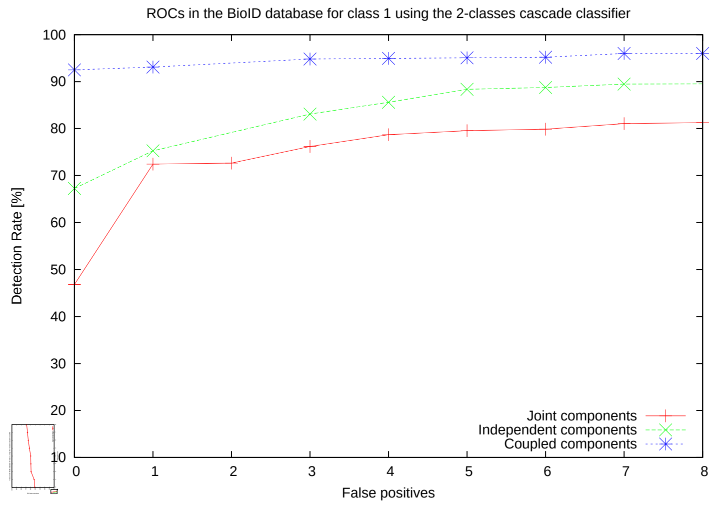
Considering the results obtained for joint components – i.e. being the one with the largest processing time, the largest training time and also the less robust one –, it will not be considered in the following sections.

Table 5.4: Performance comparison for operation point with 5 false positives; Average processing time [sec] for a 384x286 pixels image.

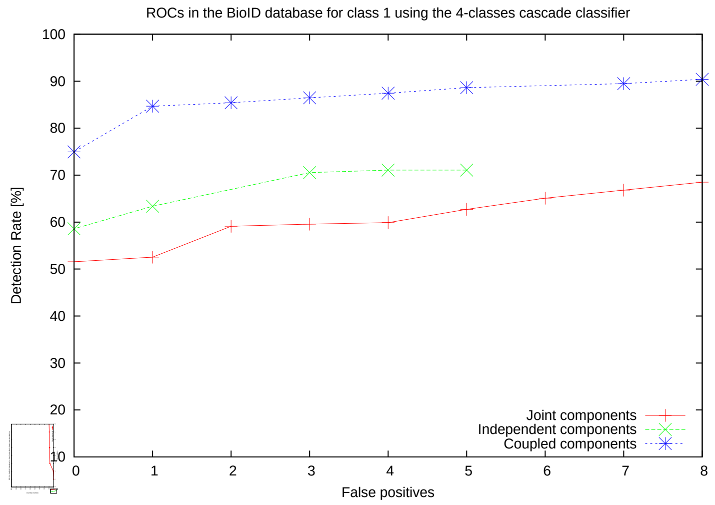
Weak classifier's training method	Detection Rate				Processing Time			
	Number of classes							
	1	2	4	8	1	2	4	8
Independent	93.56	88.36	71.07	**60.88**	0.37	0.40	0.44	4.46
Jointly trained	93.82	79.55	62.72	18.94	0.37	0.54	1.49	0.79
Coupled	**95.27**	**95.07**	**88.63**	46.94	0.35	1.0	4.29	4.99



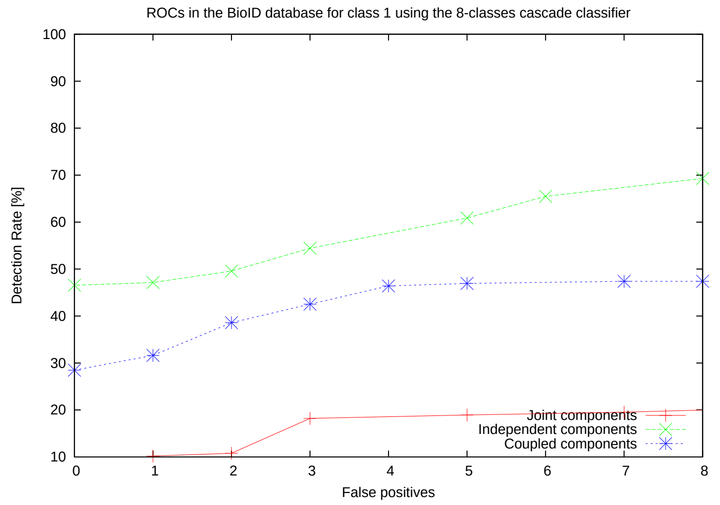
(a) 1-class



(b) 2-classes



(c) 4-classes



(d) 8-classes

Figure 5.4: ROCs for the 1-class, 2-classes, 4-classes, and 8-classes cascades used for detecting faces of class 1.

5.4 Multiclass Coarse-To-Fine Nested Cascades

In the present section we propose the use of a coarse-to-fine (CTF) search in the object target space. For this we introduce a small but importance change to the multiclass nested cascade presented in the previous section. For this we make use of a function that has a binary output and that represents a subsets of active components. At layer k of the cascade, this function is written as $\vec{\mathbf{A}}_k(\cdot)$, we refer to it as *active mask* of layer k, and is defined component-wise by:

$$\mathbf{A}_k(x,m) = \prod_{i=0}^{k} u\left(H_i(x,m)\right) = u\left(H_k(x,m)\right) \prod_{i=0}^{k-1} \mathbf{A}_i(x,m) \tag{5.13}$$

with $u(x)$ the unit step function (equal to 1 if $x \geq 0$ and 0 otherwise). 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*, we now redefine the output of a layer, k, of the CTF cascade as:

$$\vec{H}_k(x) = \left[\vec{H}_{k-1}(x) + \sum_{t=1}^{T_k} \vec{h}_{k,t}(f_{k,t}(x)) - \vec{b}_k\right] \odot \vec{\mathbf{A}}_{k-1}(x) \tag{5.14}$$

$$\text{with } \vec{H}_0(x) = \vec{0}, \text{ and } \vec{\mathbf{A}}_0(x) = \vec{1}\ ,$$

where $\odot$ is the point-wise product between two vectors (see Algorithm 6 for a formal definition).

Equation 5.14 can be interpreted as verifying the condition of the input belonging to a particular class (Equation 5.2) at each layer of the cascade, in a per class manner, and that this is done only for the subset of hypothesis that were already verified at the previous layers of the cascade. Note also that this allows to use the classifier to detect a subset of classes by setting the corresponding components in $\vec{\mathbf{A}}_0(x)$ to one and the remaining ones to zero.

One important thing that must be noticed in this definition is that in $\vec{H}_k(\cdot)$, only non-zero components of $\vec{\mathbf{A}}_{k-1}(x)$ need to be evaluated 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. This is why we called it a coarse-to-fine cascade in the target object space. Also, as in a standard cascade classifier, there is also a coarse-to-fine search on the non-object space (at each layer a sample can be discarded or passed to the next layer). Note that in the non-CTF cascade (Equation 5.12, Section 5.3) the decision about the positive classes was done at the end of the cascade, and at each layer only if all components are zero, while here is done component-wise at each layer. An example of a multiclass cascade with CTF search in the object target space is presented in Figure 5.5.

The training of the cascade remains the same (the algorithms 7, 9, 8, and 10 do not change). It is important to recall that the CTF evaluation of the multiclass cascade must be considered both during evaluation and training. This is important because each layer of the cascade must be trained using samples from a distribution as similar as possible to the one that the classifier will observe during its use.

5.4.1 Analysis & Results

Following the setup of the experiments presented in the previous section (Table 5.3, Section 5.3.2), we trained detectors for 1, 2, 4 and 8 classes with different rotations (multiples of 45 degrees). Figure 5.6 presents ROC curves for classifiers on the detection of frontal face. Each curve shows

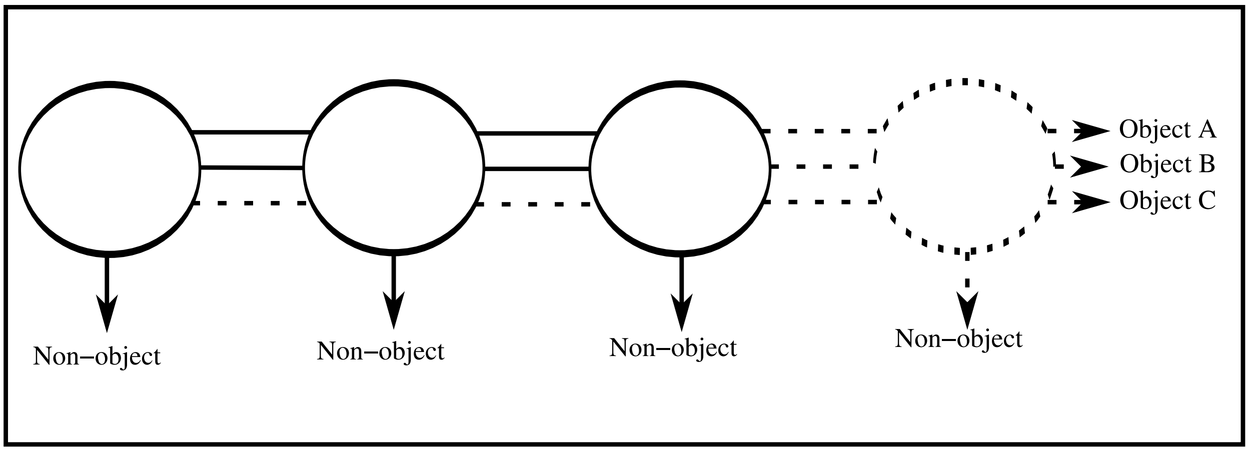


Figure 5.5: CTF multiclass cascade classifier structure. Example of a multiclass cascade consisting of 4 layers, with boosted classifier of 3 components, and where the output encodes 3 classes. In the example the dashed circles indicates layers not being evaluated, and the dashed lines indicate inactive components during the evaluation of the cascade for a particular windows being classified.

the detection results when considering a different number of classes, and applied on the BioID dataset. In these experiments the detection results with independent and coupled components were very similar, therefore the results presented here are only the ones obtained using independent components. Figures 5.8 and 5.9 present examples of detection results obtained with the 8-classes detector.

As it can be observed from Figure 5.6, the performances decreases only by a small factor, from 92% (1-class cascade) to 88% (8-classes cascade) for 5 FP when increasing the number of classes. These results are clearly better than the ones by the non-CTF cascade (Figure 5.4 (d)).

In addition, we evaluated the processing and the training time. In this case, the training of a 8-classes cascade took only 13 times more than the one class cascade (2.6 hours vs 34.3 hours on a 3GHz Pentium 4 computer)[g]. To evaluate the processing time, we took a large and complex image (2816x2112 pixels, with 11 faces and complex background). This image was processed several times using the trained multiclass detectors. The average processing time is presented in Figure 5.7. In that figure we also present: (a) the processing time of running in parallel *n* cascades (trained for 1-class), and (b) the processing time of running the multiclass (8-classes) classifier but used only to detect a subset of the classes (by setting $\vec{\mathbf{A}}_0 \in \{0,1\}^M$ to a value different than $\vec{1}$). From Figure 5.7 several things can be observed:

- The processing time of the multiclass cascades (*n*-classes) grows close to a logarithmic function, with a regression value of $R^2 = 0.99$ (a linear regression gives $R^2 = 0.94$). This time is much faster that running *n* multiclass cascades.
- When the 8-classes cascade is used to detected only a subset of the objects, the processing time only grows linearly with the number of classes.

Table 5.5 presents similar results. In this case images of 320x240 pixel were processed and the time averaged. Again we can be observe a logarithmic grow of the processing time, for example

[g]Remember that not only the number of classes increases, but also the number of training examples (Section 5.3.2). See also Table 5.2

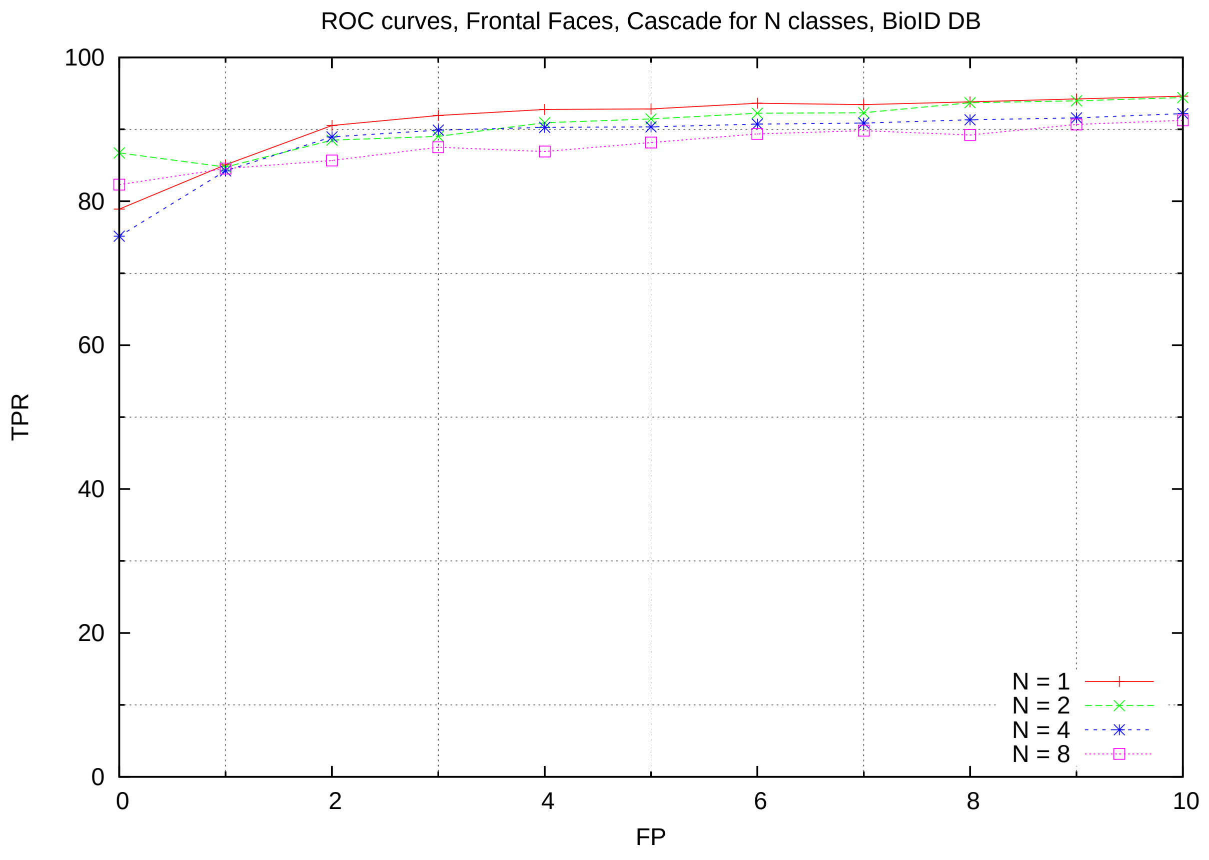


Figure 5.6: Multiclass nested cascade versus number of classes when using CTF search in the object space

the processing time of the 1-class cascades is only three times slower compared to the 8-classes cascade.

Table 5.5: Average processing time [sec] of a CTF multiclass cascade on images of 320x240 pixel size.

Weak classifier's training method	Number of classes			
	1	2	4	8
Processing time	0.22	0.33	0.60	0.67
Ratio of processing time: n-classes to 1-class	-	1.5	2.8	3.04

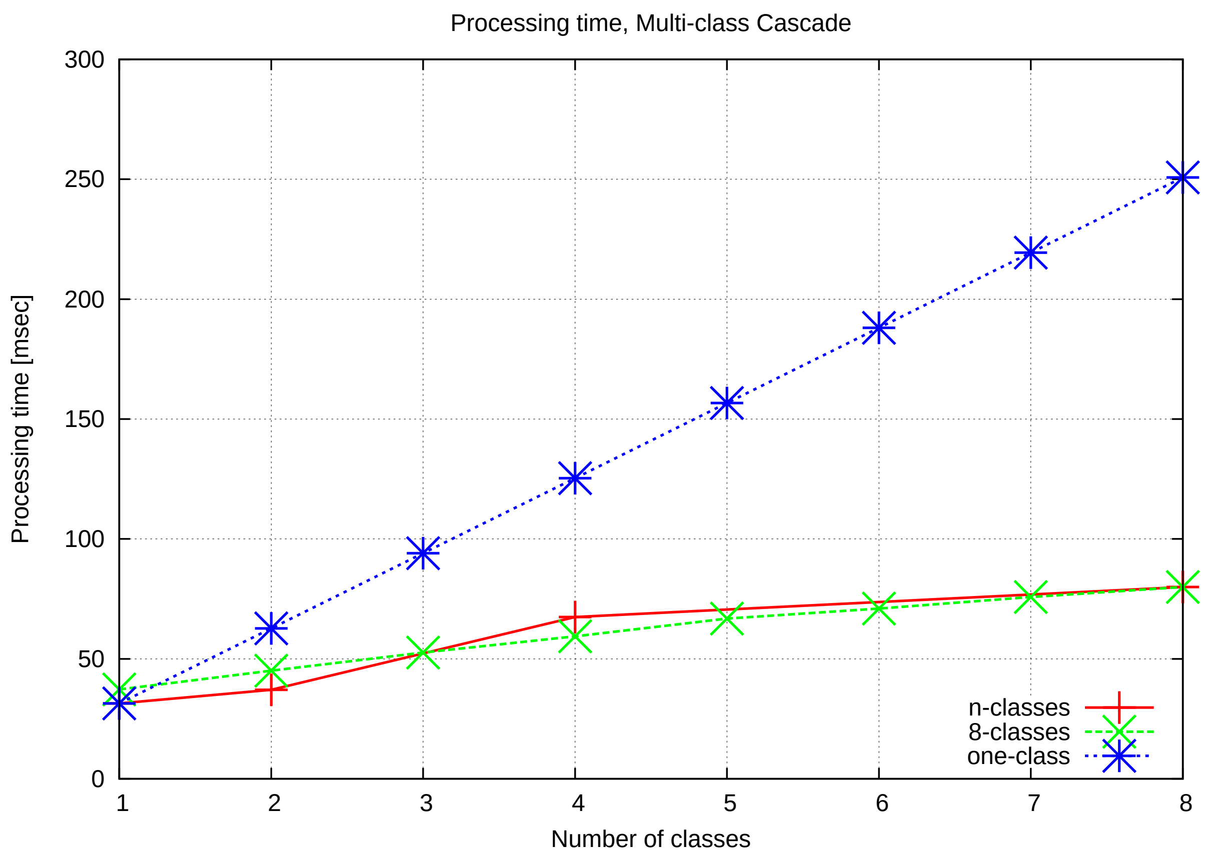


Figure 5.7: Processing time of multi-class cascade. *n-classes*: processing time of cascades trained for that specific number of classes (1,2,4,8). *8-classes*: processing time of a 8-classes cascade disabling some components. *one-class*: processing time of running *n* times a one-class cascade (estimation based on a 1-class cascade).

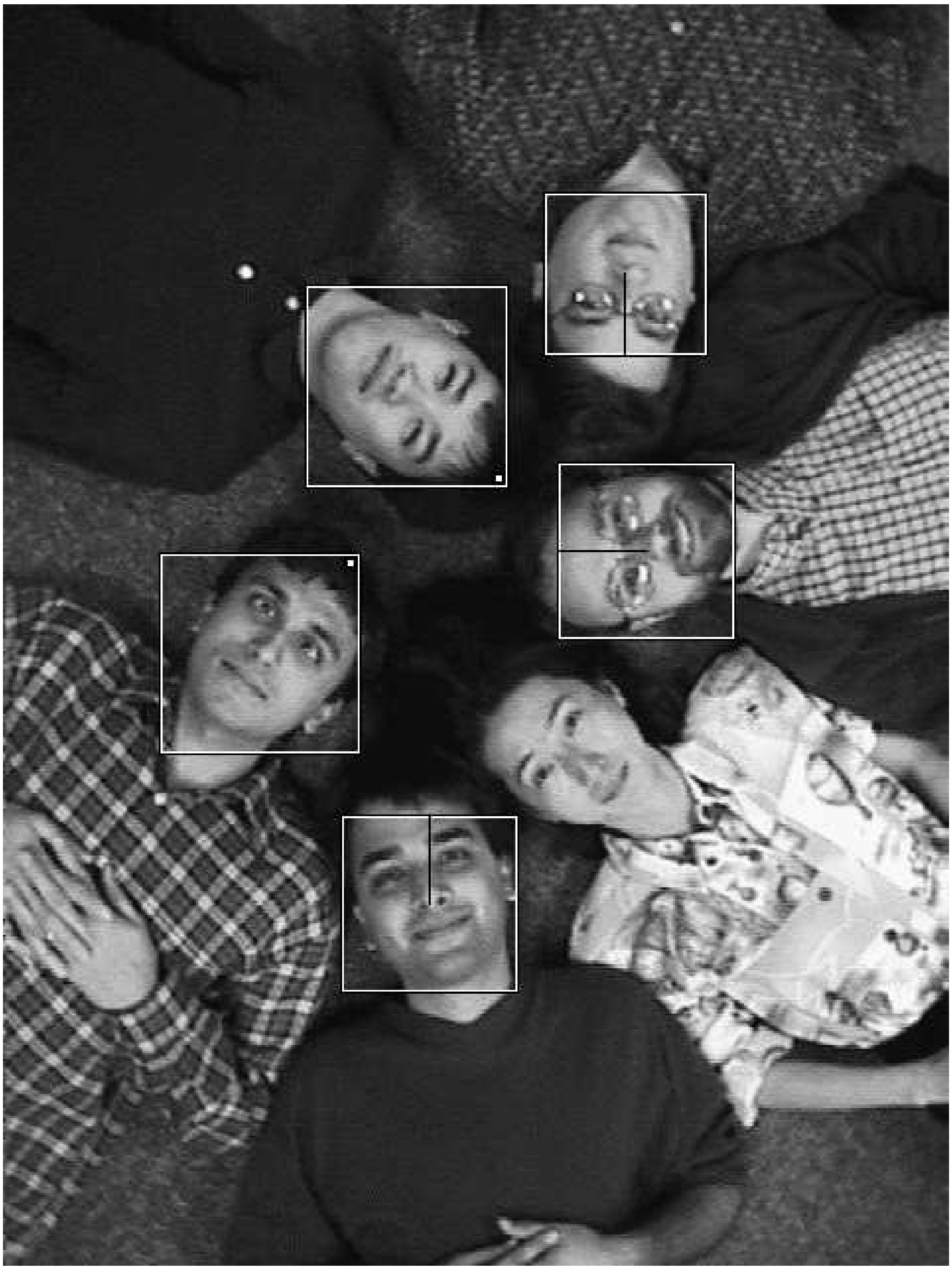

(a) Detected faces: 5 out of 6; False positives: 0; Image size: 480x640

Figure 5.8: Detection result examples of the 8-classes CTF cascade detector. The line/small-rectangle indicates the predicted class by indicating the position of the forehead.

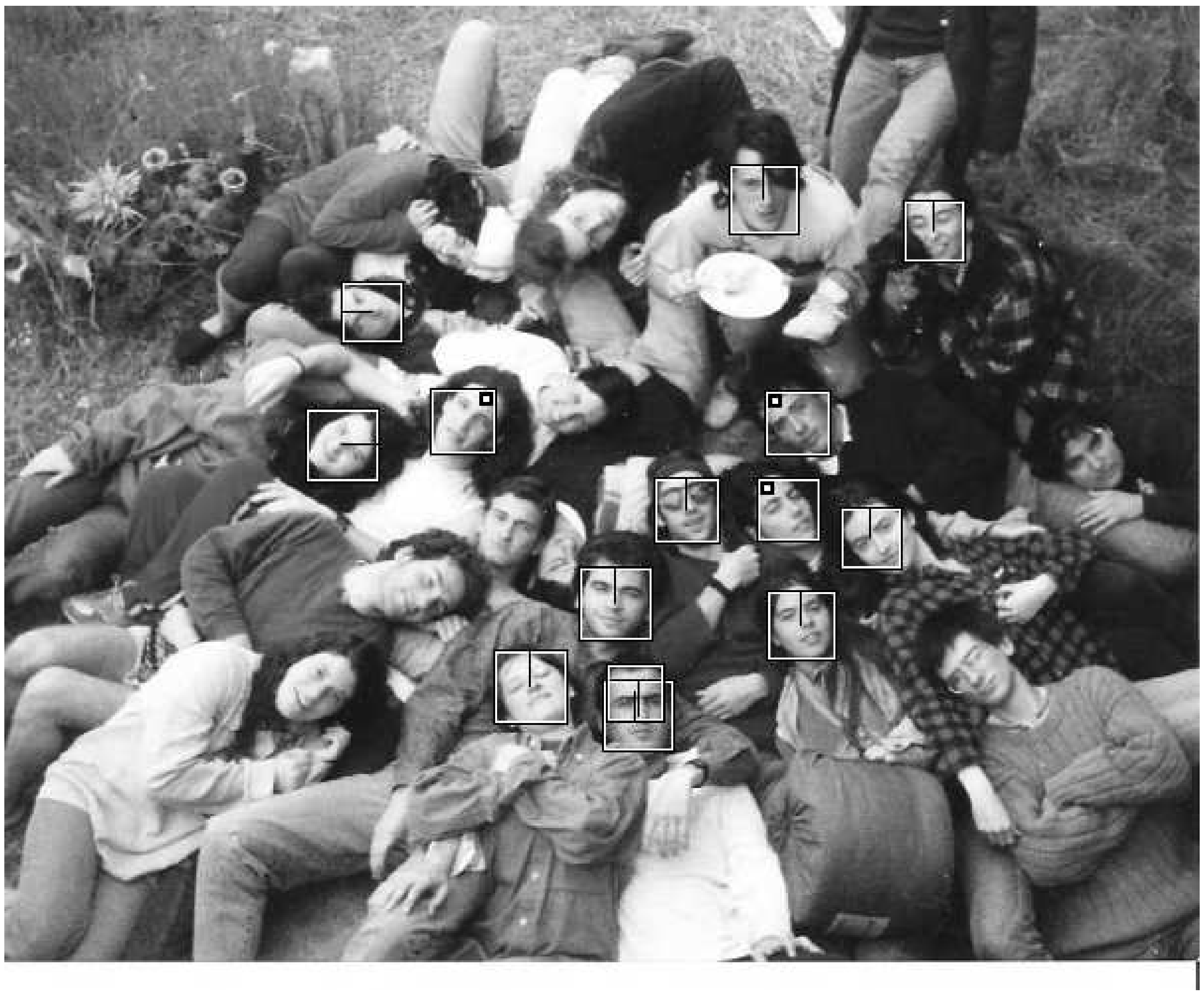

(a) Detected faces: 13 out of 22 (note that some faces have rotations that do not match the classes of the detector); False positives: 1; Image size: 604x497.

Figure 5.9: Detection result examples of the 8-classes CTF cascade detector. The line/small-rectangle indicates the predicted class by indicating the position of the forehead.

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

In the present section the proposed nested coarse-to-fine tree classifier, so called *TCAS*, is introduced. The main advantages of a *TCAS* are its ability to handle problems with many different classes obtaining high accuracy and high processing speed.

The multiclass nested cascade presented in the previous section presents good results, but it has one problem. In the cases when many object-classes were considered, when some of the classes were "different" to each other, and when very high accuracy per layer was required, the training did not converge or the obtained classifier was very slow. In other words, we were not able to obtain high detection rates and low false positives rates per layer without increasing considerably the number of weak classifiers and in some case the training of the last layers of the cascade did not converge. This problem appeared after already having added at least a few layers to the multiclass cascade.

This has to do mainly with the complexity of building classifiers that can discriminate between the object class(es) and the background (the non-classes), which is a problem that becomes more and more important as layers are added to the cascade. This problem is particularly important in the multiclass case; after some layers have been added to the cascade, the bootstrapped "non-objects" become less "similar" to each other because each "negative" example is "negative" for a different subset of objects, i.e. each "negative" is only "negative" for only a few classes. More precisely, this means that it becomes difficult to find features that can be use to discriminate all objects classes from "all" background(s).

In order to solve this problem, a coarse-to-fine tree classifier structure is proposed, classifier that we refer to as *TCAS*. This tree structure consist of a *nested* tree – nested because the output of a node depends on the output of its ancestors–, where each node of the tree contains a nested multiclass cascade. The basic ideas are that each sibling of a node has to deal only with a subgroup of the classes that its ancestors had to deal with, that the nodes of the trees are evaluated in a nested fashion, and that more than one node sibling of the tree can be followed.

5.5.1 Tree Classifier Structure

A *TCAS* classifier, $\mathbb{T}$, corresponds to a directed tree, with each node having a variable number of siblings. A node $\mathbb{N}$ of $\mathbb{T}$ has $n_{\mathbb{N}}$ siblings, $\{\mathbb{N}_s\}_{s=\{1,\ldots n_{\mathbb{N}}\}}$, and consist of:

- a multiclass classifier $\vec{H}_{\mathbb{N}}$ (in our case a multiclass nested cascade),
- a mask $\vec{\mathbf{A}}_{\mathbb{N}} \in \{0,1\}^M$,
- and a "pointer", $p_{\mathbb{N}}$, to its direct ancestor.

An example of a *TCAS* structure is presented in Figure 5.10, and some auxiliary functions and notation are introduced in Algorithm 11.

Every node, $\mathbb{N}$, of a *TCAS* has a nested structure, i.e its output depends on the output of its ancestors, and is defined as:

$$\vec{H}_{\mathbb{N}}(x) = \vec{H}_{p_{\mathbb{N}}}(x) \odot \vec{\mathbf{A}}_{\mathbb{N}} + \vec{H}_{\mathbb{N}}(x), \tag{5.15}$$

with $\vec{H}_{p_{\mathbb{N}}}(x)$ the output of the ancestor of $\mathbb{N}$, $p_{\mathbb{N}}$. Note that if $p_{\mathbb{N}}$ is the root of the tree then $\vec{H}_{p_{\mathbb{N}}}(\cdot) = \vec{0}$. It is important to note that only non-zero components of $\vec{\mathbf{A}}_{\mathbb{N}}$ need to be evaluated in $\vec{H}_{\mathbb{N}}(x)$, i.e. the coarse-to-fine evaluation in the object target space – already introduced for the multiclass cascade – is also used here, which allows to maintain a efficient evaluation.

The mask $\vec{\mathbf{A}}_{\mathbb{N}}$ indicates which components/classes of the classifier are considered at the current node. Thanks to this, all nodes of the tree have an output with the same number of components, but at each node only a subset of the components is active.[h] Regarding this, one important restriction is made (recall that a mask $\vec{\mathbf{A}} \in \{0,1\}^M$):

$$\vec{\mathbf{A}}_{\mathbb{N}} = \sum_{s=Siblings(\mathbb{N})} \vec{\mathbf{A}}_s \tag{5.16}$$

which means that the (binary) mask for any two siblings of node $\mathbb{N}$, $\mathbb{N}_i$ and $\mathbb{N}_j$, with $i \neq j$, it holds that $\vec{\mathbf{A}}_{\mathbb{N}_i} \odot \vec{\mathbf{A}}_{\mathbb{N}_j} = 0$. This restriction allows to simplify the evaluation of the output of the tree by allowing an efficient recursive implementation and also simplifies the training process, because two nodes that are in different branches of the tree do not depend on each other (see details of the training in Section 5.5.2).

Given that the output of all nodes have the same dimension and 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 $\mathbb{N}_{(1)}, \ldots, \mathbb{N}_{(n_{leafs})}$

$$\vec{H}_{\mathbb{T}}(x) = \sum_{j=1,\ldots,n_{leafs}} \vec{H}_{\mathbb{N}_{(j)}}(x) \tag{5.17}$$

nevertheless this can be implemented in an efficient way thanks to the restriction of equation (5.16). Algorithm EVALTREE (see Algorithm 12) presents an efficient recursive implementation of the evaluation of a *TCAS*. Some auxiliary functions are presented in Algorithm 11. This efficient implementation is possible thanks to the fact that, a node $\mathbb{N}$, can only modify the output of a subset of components that do not overlaps with the output of any other node that is not one of its ancestors.

Note that if only one branch of the tree is followed, the evaluation is identical to that of a multiclass nested cascade proposed in the previous section.

[h] In a more general setting, the mask $\vec{\mathbf{A}}_{\mathbb{N}}$ could be replaced by a matrix (and $\odot$ by a matrix product), so that the dimensions of a node and each of its siblings could be different. This would allow to have a nested structure and, for example, to group classes in the first nodes of the tree, or to have a nested structure were the nesting depends on more than one components of the ancestor.

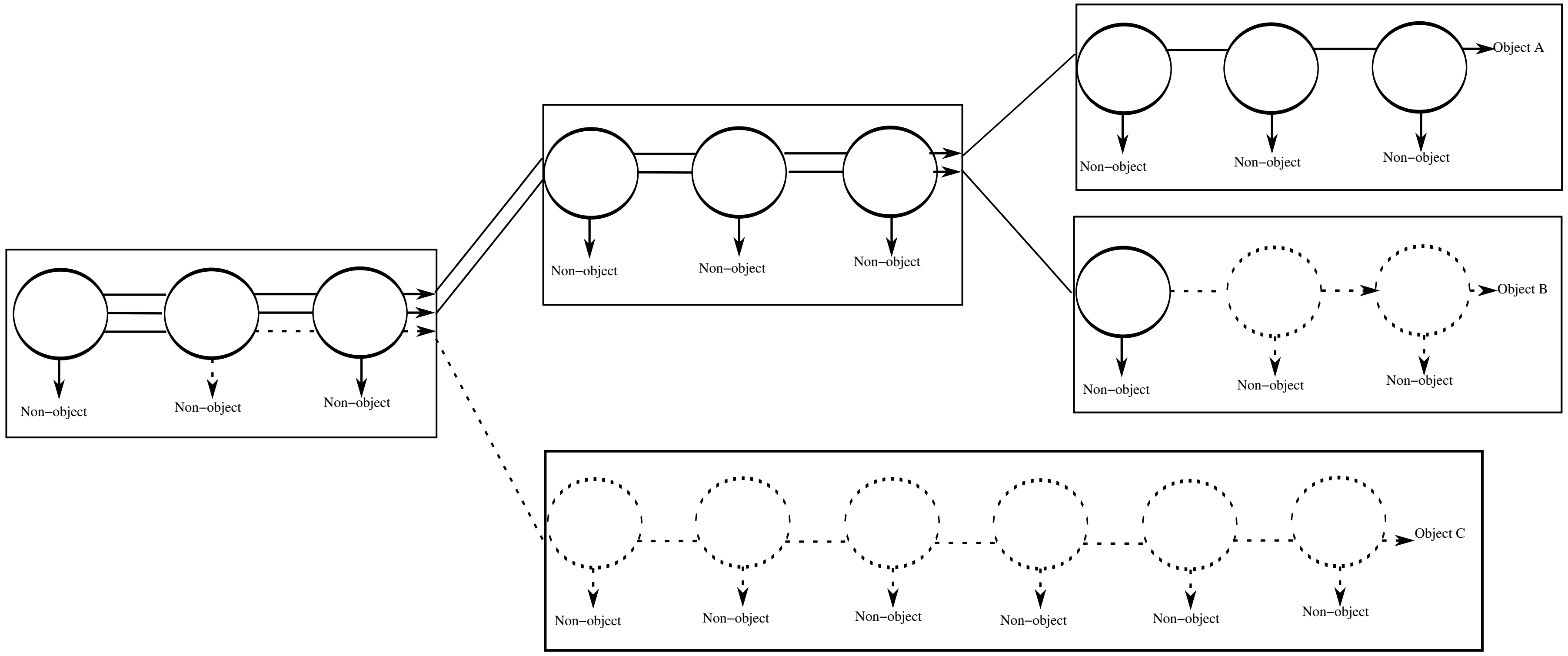


Figure 5.10: Example of a *TCAS* structure. Example of a multiclass *TCAS* for a 3-classes detection problem. The tree consisting of five nodes, where four nodes have a multiclass with three layers and one node has a cascade of 6 layers. In the example, the dashed circles indicates layers not being evaluated, and dashed lines indicate inactive components during the evaluation of the tree for a particular windows being classified.

Algorithm 11 *TCAS*- Auxiliary Functions, Definitions & Notation

```
1: The Tree Node Structure, $\mathbb{N}$, consist of:
   $n_{\mathbb{N}}$ siblings $\{\mathbb{N}_s\}_{s=\{1,\ldots n_{\mathbb{N}}\}}$
   $\vec{H}_{\mathbb{N}}(x)$: a classifier (a CTF multiclass nested cascade)
   $\vec{\mathbf{A}}_{\mathbb{N}}$: a binary vector (mask) indicating the components of $\vec{H}_{\mathbb{N}}(x)$ active at the node
   $p_{\mathbb{N}}$: pointer to its direct ancestor
2: $GetCascade(\mathbb{N})$ returns the classifier (cascade) of node $\mathbb{N}$.
3: $Ancestor(\mathbb{N})$ returns the direct ancestor of $\mathbb{N}$, FALSE if root.
4: $Mask(\mathbb{N})$ returns the mask of node $\mathbb{N}$.
5: $Siblings(\mathbb{N})$ returns the siblings of node $\mathbb{N}$.
6: $newSibling(\mathbb{N})$ creates an empty node and adds and edge to $\mathbb{N}$ (new sibling).
7: $Binary(\vec{V})$ returns a vector $\in \{0,1\}^M$, which is, per component $m$, equal to:
   $$\left(Binary(\vec{V})\right)_m = u\left((\vec{V})_m\right) = \begin{cases} 1 & \text{if } (\vec{V})_m > 0) \\ 0 & \sim \end{cases}$$
8: Stack handling
   $Push(ST,[N,A])$ adds $[N,A]$ to the top of the stack $ST$
   $Pop(ST)$ removes and returns the top element of the stack $ST$ (and FALSE if empty).
```

5.5.2 Tree Classifier Training

The training of a *TCAS* is based on the methods already developed to train multiclass nested cascades, and is presented in the algorithm TRAINTREE (see Algorithm 13). The training is a bottom-up procedure, which recursively adds nodes to the tree starting from the root node. Nodes are added by taking into account the multiclass nested cascade that can be built by traversing the ancestors of the node (line 4, using algorithm FLATTENING), and the training examples that have a target region defined by the active mask of the current node (lines 5 - 7). A node is trained and added (line 8, using algorithm TRAINNODE) considering the relevant training examples and cascade. Later, if need, the components of a node are grouped (line 10), and the new siblings are initialized and added to the stack of nodes to be trained (lines 9 - 14). The training of the tree stops when the stack is empty (line 3), event that happens because at some point no sibling was added to the nodes. There are two reasons for not adding more nodes to a branch of the tree, either the branch of the tree already fulfils the required false positive rate (line 9) or the function GROUPCOMPONENTS returns an empty set. It is important to note that thanks to this, the number of layers to be added is not predefined, and it decided based on the complexity of the learning problem (number of weak classifiers added to the last layer). This means that the depth of the branches of the tree is variable.

The FLATTENING algorithm (see Algorithm 14) constructs a cascade associated to a branch of the tree starting from a leaf node and going backwards up to the root tree (line 3). The cascade is built considering only the components that are active at the leaf node (line 4).

The training of a node is done by the algorithm TRAINNODE (see Algorithm 15), which adds new layers to the input cascade considering the active components (lines 1-3). Afterwards (lines 4-7) it obtains the newly trained layers (which are the layers to be set as the cascade of the new node), but in some cases the last layer (line 5-7) is removed. The removal of the last layer (line 6) is needed when the training of the cascade did not converged for the last layer ($t \geq T_{max}$), and this

Algorithm 12 EVALTREE($\mathbb{T}, \vec{\mathbf{A}}_0, x$)

```
Input: $\mathbb{T}$: tree root
Input: $\vec{\mathbf{A}}_0$: Initial mask (components to be considered by the classifier)
Input: $x$: window (image patch) to be classified
 1: $\vec{O}(x) \leftarrow \vec{0}$ // Initialize output
 2: $Push\left(ST, [\mathbb{T}, \vec{\mathbf{A}}_0 \odot Mask(\mathbb{T})]\right)$ // Initialize stack with the root node
 3: while $\left([\mathbb{N}, \vec{\mathbf{A}}] \leftarrow Pop(ST)\right)$ do
 4:     $\vec{H}_t \leftarrow GetCascade(\mathbb{N})$
 5:     $\vec{O}(x) \leftarrow \vec{O}(x) + \vec{\mathbf{A}} \odot \vec{H}_t(x)$
 6:     $\vec{\mathbf{A}}_t \leftarrow \vec{\mathbf{A}} \odot Binary(\vec{H}(x))$
 7:     if $\vec{\mathbf{A}}_t \neq 0$ then
 8:         for each $\mathbb{N}_s \in Siblings(\mathbb{N})$ do
 9:             if $\vec{\mathbf{A}}_t \odot Mask(\mathbb{N}_s) \neq 0$ then
10:                 $Push\left(ST, [\mathbb{N}_s, \vec{\mathbf{A}}_t \odot Mask(\mathbb{N}_s)]\right)$
11:             end if
12:         end for
13:     end if
14: end while
Output: $\vec{O}(x)$
    Note: From the training we have that $\sum_{\mathbb{N}_S = Siblings(\mathbb{N})} Mask(\mathbb{N}_s) = Mask(\mathbb{N})$
    Note: At line 5, only non-zero components of $\vec{\mathbf{A}}$ need to be evaluated in $\vec{H}_t(x)$
```

removal is applied only in the cases when the new cascade has at least one layer ($n_{tail} > 1$).

Grouping of the Components (classes)

At lines 9-14 of algorithm TRAINTREE (see Algorithm 13), in the method GROUPCOMPONENTS, the active components of a node are grouped. Each of these groups is assigned to a siblings of that node. This can be implemented in many different ways, but this procedure must fulfill two conditions. The first one is equation (5.16), which means that the groups must have an empty intersection and that the union of the groups is equal to the input set. The second condition is that if the input mask has only one active component, it must return the empty set, condition that is used to stop adding nodes to that branch, i.e. the node will not have any sibling.

The grouping procedure can be implemented in many ways, and the main point is deciding which criteria to use to group the classes, including how many groups will be obtained. It is important to recall that the grouping to be done is a grouping of the classes/components and not a grouping of the examples (as done in [Wu and Nevatia, 2007]).

The results presented later in this chapter only consider the case when the groups are predefined a priori, before starting the training procedure (as done by [Huang et al., 2007]). We have analysed and tested other cases, including random balanced partitions and partitions based on the "similarity" between the classes (measured by the distribution of the position of the weak classifiers). This last procedure showed good results, but these results and methods are to be presented elsewhere.

5.5.3 Comparison with Related Tree Classifiers

The *TCAS* structure shares some similarities with existing classifiers: Width-First-Search (WFS) trees [Huang et al., 2007], Cluster-Boosted-Trees (CBT) [Wu and Nevatia, 2007], and Alternating Decision Trees (ADT) [Freund and Mason, 1999], but the structure of the classifier and the training procedures are different. The differences are briefly outlined in the following.

The Cluster Boosted Tree (CBT) proposed in [Wu and Nevatia, 2007] is designed only for multiview detection problems, where the partition of one class of objects in subviews (called channels in [Wu and Nevatia, 2007]) is learnt. The obtained structure is very similar to the one obtained here, with the main differences being: (1) the depth of the tree is the same for all branches, and (2) each node of the tree contains a single "layer", while in our case it contains a cascade. These differences reflect in that in the tree proposed here, there is a coarse-to-fine search in the object space within each node of the tree, and the depth of each branch of the tree is variable, with some branches being shorter if the classes associated to it are "simpler". In addition, the work proposed in [Wu and Nevatia, 2007] is meant for multiview object detection problems, where "sub-classes" are learnt, while the methods proposed here are meant for both multiclass and multiview problems. Therefore both methods can be seen as complementary[i].

In [Huang et al., 2007] a Width-First-Search (WFS) tree is proposed. The WSF tree structure is similar to the *TCAS*, but the main differences are that: (1) the WSF-tree is not nested, and (2) the WFS tree does not use CTF multiclass cascades – they use cascades but no detail is given about their training or structure. The main similarities are the use of Vectorboost and the use of a *WFS* search. The term WFS refers to the fact that at each node, any number of siblings can be followed when traversing the tree.

The *TCAS* tree structure also shares some similarities with Alternating Decision Trees (ADT) [Freund and Mason, 1999]. ADT trees are trained using a boosting procedure and to the best of our

[i]A future line of research could consist of combining both methods.

knowledge, ADTs have not been used in object detection problems. The main similarity between ADT and the *TCAS* is that in an ADT the output of a leaf depends on all nodes of the branch the leafs belongs to (i.e. is "nested"), but in addition the output may also depend on nodes of other branches. ADT 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. Another important difference is that in an ADT, during training, a node (a weak classifier) can be added to any leaf, which requires to maintain "global" weights, while in our case the weights used during boosting are "local" to the node being trained.

Algorithm 13 TRAINTREE$(S, B, F, F_C, D, PV, NV, T_{max})$

```
Input: $S = \{(x_i, \vec{a}_i)\}_{i=1,\ldots,n}$ Set of positive examples
Input: $B$: Bootstrapp set
Input: $F$: Maximum false positive rate per node, and $F_C$: global false positive rate
Input: $D$: Minimum detection rate per node
Input: $PV$ / $NV$: Positive/Negative Validation Set
Input: $T_{max}$ maximum number of weak classifiers of a layer of the cascade.
 1: $\mathbb{T} \leftarrow \{\emptyset\}$
 2: $Push(ST, [\mathbb{T}, \vec{1}])$
 3: while $([\mathbb{N}, \vec{\mathbf{A}}] \leftarrow Pop(ST))$ do
 4:     $\vec{H}_t \leftarrow$ FLATTENING$(\mathbb{N})$
 5:     $S_t \leftarrow filterExamples(\vec{\mathbf{A}}, S)$
 6:     $PV_t \leftarrow filterExamples(\vec{\mathbf{A}}, PV)$
 7:     $NV_t \leftarrow filterExamples(\vec{\mathbf{A}}, NV)$
 8:     $\vec{H}_{\mathbb{N}} \leftarrow$ TRAINNODE$(\vec{H}_t, \vec{\mathbf{A}}, S_t, B, F, F_C, D, PV_t, NV_t, T_{max})$
 9:     if $fpr\left(\vec{H}_t + \vec{H}_{\mathbb{N}} \odot \vec{\mathbf{A}}\right) \geq F_C$ then
10:         $\mathbb{G} \leftarrow GroupComponents(\vec{\mathbf{A}})$ // This procedure fulfils that $\sum_{\vec{\mathbf{g}} \in \mathbb{G}} \vec{\mathbf{g}} = \vec{\mathbf{A}}$
11:         for each $\vec{\mathbf{g}} \in \mathbb{G}$ do
12:             $Push(ST, [newSibling(\mathbb{N}), \vec{\mathbf{g}}])$
13:         end for
14:     end if
15: end while
Output: $\mathbb{T}$
    $S_t \leftarrow filterExamples(\vec{\mathbf{A}}, S)$: returns $S_t = \left\{(x_i, \vec{a}_i) \in S \mid \left(\vec{a}_i \odot \vec{\mathbf{A}}\right) \neq \vec{0}\right\}$
```

Algorithm 14 FLATTENING($\mathbb{N}$)

// Get the cascade associated to a branch of the tree starting from a leaf node $\mathbb{N}$

```
Input: $\mathbb{N}$: tree node
1: $\vec{H}(\cdot) \leftarrow \vec{0}$
2: $p \leftarrow \mathbb{N}$
3: while $(p \leftarrow Ancestor(p))$ do
4:     $\vec{H}_t(\cdot) \leftarrow Mask(p) \odot GetCascade(p)$
5:     $\vec{H}(\cdot) \leftarrow \vec{H}(\cdot) \cup \vec{H}_t(\cdot)$
6: end while
Output: $\vec{H}$
```

Algorithm 15 TRAINNODE($\vec{H}, \vec{\mathbf{A}}, S, B, F, F_C, D, PV, NV, T_{max}$)

```
Input: $\vec{H}$: Cascade classifier
Input: $\vec{\mathbf{A}}$: Mask indicating the active components of $\vec{H}$ at that node
Input: $S$: Positive Train Set
Input: $B$: Bootstrapp set
Input: $F$: Maximum false positive rate per node, and $F_C$: global false positive rate
Input: $D$: Minimum detection rate per node
Input: $PV$ / $NV$: Positive/Negative Validation Set
Input: $T_{max}$ maximum number of weak classifiers of a layer of the cascade.
1: $\vec{H}_{init} \leftarrow \vec{H} \odot \vec{\mathbf{A}}$
2: $\vec{H}_t \leftarrow$ TRAINCASCADEMC($\vec{H}_{init}, S, B, F, F_C, D, PV, NV, T_{max}$)
3: $n_{tail} \leftarrow nLayers(\vec{H}_t) - nLayers(\vec{H}_{init})$
4: $\vec{H}_{tail} \leftarrow lastLayers(\vec{H}_t, n_{tail})$
5: if $\Big((n_{tail} > 1) \wedge \Big(sizeBoosted\Big(lastLayers(\vec{H}_{tail}, 1)\Big) \geq T_{max}\Big)\Big)$ then
6:     $\vec{H}_{tail} \leftarrow firstLayers(\vec{H}_{tail}, n_{tail} - 1)$ // Remove last layer
7: end if
Output: $\vec{H}_{tail}$ // returns the layers that were added to the tail of the cascade
    $nLayers(\vec{H})$: returns the number of layers of the cascade $\vec{H}$
    $sizeBoosted(\vec{H})$: returns the size of the boosted classifier
    $firstLayers(\vec{H}, n)$: returns a cascade consisting of the first $n$ layers of $\vec{H}$ $lastLayers(\vec{H}, n)$:
    returns a cascade consisting of the last $n$ layers of $\vec{H}$
```

5.6 Evaluation & Results

In the present section, results on two different settings are presented: multiview object detection, and multiclass object detection. In both detection problems we focus on evaluating the performance (detection rates, false positive rates, and processing times) of *TCAS* compared to parallel cascades. On these problems, and given their complexity, it was not possible to obtain multiclass cascades for high true positive rates and low false positive rates per layer, therefore results are only presented for *TCAS* and multiple cascades running in parallel.

To evaluate the proposed methods in a multiview detection setting, we consider the problem of detecting faces with rotation-in-plane (RIP), where a comparison to state of the art methods on a standard database is presented. The evaluation in the multiclass setting is done in the problem of detecting frontal faces, left fist hands and right fist hands.

For building the *TCAS*s we consider the use of multiclass weak classifiers trained using coupled components (*TCAS*-c), independent components (*TCAS*-i), and coupled components in the first levels of the tree and independent components in the following ones (*TCAS*-ci). In addition, the use of different combinations of classes and different number of classes is analyzed.

5.6.1 Rotation-In-Plane (RIP) Invariant Multiview Face Detection

In order to analyse and evaluate the proposed multiclass *TCAS* classifier on a multiview setting, we consider the problem of rotation-in-plane (RIP) invariant face detection. We perform the evaluation and analysis on the CMU Rotated set [Rowley et al., 1998], which is the standard database used on this problem. The CMU Rotated set consist of 50 images containing in total 223 faces. The faces present only in-plane rotations, i.e., rotations with respect to the camera plane. The rotations are in the whole range, going from 0 to 360 degrees. The database has two important particularities that must be mentioned: *i*) there is a very large image (2615x1986 pixels) that contains more than half of the faces of the database (134 faces), with all faces having a rotation close to 30 degrees (the image was rotated in 30 degrees, see Figure 5.16), and *ii*) there are 12 low quality images (very noisy images of scanned newspapers, see Figure 5.14 (b) for an example), images which contain 25 faces in total (11% of the database's faces) and some of these faces could not be detected by any of the built classifiers.

To perform the analysis, in all cases, the classes are defined as view-ranges of 18 degrees. For example, to cover the range $[-45,45)$, five classes are considered, covering the ranges $[-45,-27)$, $[-27,-9)$, $[-9,9)$, $[9,27)$, and $[27,45)$. Then a 5-classes *TCAS* is trained (or five 1-class cascades are trained). In this way the same training sets are use to train the *TCAS* and the parallel cascades. In order to cover the full range $[0,360)$, the input image is rotated 4 times, in 0, 90, 180, and 270 degrees, and the classifiers are applied to each of these images (note that the classifier could be rotated instead of the image). In all experiments presented in the following, to obtain the final detections, the output from all leaf nodes (or of the cascades) is just considered as a face and the range (class) is omitted. In this way, the same post-processing done in Chapter 4 is applied to the windows in the overlapping post-processing module.

The training set for each of these ranges was built using the ISL WebImage dataset[j], taking the frontal and quasi frontal faces (800 labeled faces) and rotating them randomly (uniform distribution) in the corresponding range. Afterwards, each of these images was post-processed by applying random translations and scale changes. This procedure was repeated to obtain in total 1600 training

[j]Available by request at http://www.ecse.rpi.edu/ cvr/database/database.html, Intelligent System Lab, Rensselear Polytechnic Institute.

examples per range. The number of negative examples collected during bootstrapping was 3200 per class (view-range), i.e., when building a node of a *TCAS* that has 5 classes, 16000 ($5*3200$) negative examples and 8000 ($5*1600$) positive examples are used and when building a 1-class cascades, 3200 negative examples and 1600 positives are used.

All classifiers are trained with exactly the same parameters (*fpr* per layer, *tpr* per class, per layer, features, etc). Rotated and non-rotated rectangular features were used in the first two layers, while in later layers mLBP features were considered. The *fpr* per layer was set to 0.35, the *tpr* per layer per class was set to 0.9995, and the target *fpr* was 10^{-6}. As in previous experiments, 20% of the features are considered at each iteration of Adaboost when rectangular features are used. When training the *TCAS*, the triggering of the splitting of a node is done when 50 weak classifiers are added ($T_{max} = 50$) and at most three siblings per node were considered (they were group by adjacency of their corresponding view-range).

Weak Classifier Training Method

As previously mentioned, three variants of the *TCAS* are considered: (a) the nodes of the tree are trained using coupled components (*TCAS*-c), (b) the nodes are trained using independent components (*TCAS*-i), and (c) the nodes in the first two levels of the tree are trained with coupled components and in deeper levels independent components are used (*TCAS*-ci). This analysis is performed using 5 classes. Note that in the case of *TCAS* trained using coupled components (*TCAS*-c) the layers of the tree having only one active class are trained using the analytical solution (the same procedure used to train the 1-class cascades), and not with the quasi-Newton method (see Section 5.2.3), because the training time is shorter.

Figure 5.11 presents the obtained ROC curves and Table 5.6 presents the processing and training time of the obtained classifiers, as well as information of their structure.

In terms of the processing time (Table 5.6)[k], the *TCAS*-c detector is approximately 1.17 times faster than *TCAS*-i while being 1.22 times faster than the *TCAS*-ci. The *TCAS*-ci is approximately 1.1 times faster than the *TCAS*-i.

The best performing variant, in terms of classification rates, is *TCAS*-ci, followed by *TCAS*-i and the *TCAS*-c. As it can be observed *TCAS*-c has a detection rate 1 to 2 percentage points lower than the *TCAS*-i and the *TCAS*-ci for false positive larger than 35. For a lower number of false positives, the difference becomes much larger, going up to 4 percentage points when is compared to *TCAS*-ci. On the contrary, *TCAS*-ci and *TCAS*-i have similar detection rates for false positives larger than 35, but for lower false positives, *TCAS*-ci has a clear better performance, in particular for low false positives, with differences up to 4 percentage points.

In terms of training time, the *TCAS*-i is the one that takes the largest (187 minutes), followed by the *TCAS*-ci (176 minutes), the *TCAS*-c (161 minutes).[l] The training and processing times are directly related to the number of weak classifiers that must be evaluated in the first layer. The *TCAS*-ci and the *TCAS*-c only contain 17 weak classifiers in the first layer, while the *TCAS*-i contains 27.

On the contrary, the sum of the total number of weak classifiers is 3915 for the *TCAS*-i, 2301 for the *TCAS*-c, and 3913 for the *TCAS*-ci. Note that the number of weak classifiers becomes important when there are memory restrictions, case were the use of coupled components requires the fewest number of weak classifiers. Recall that couple components use fewer parameters by design and here, in addition, requires fewer weak classifiers without presenting a large difference

[k]The presented processing times were obtained on a Intel Core 2 Duo CPU T7500 at 2.20GHz running Linux.

[l]These times were obtained on a Intel Core i7 CPU 920 at 2.67GHz running Linux and taking advantage of the 8 virtual cores. They training time without using any parallelism is about 8 times longer.

in the performance and being the fastest one. Therefore, in applications with memory restrictions it could be better to use *TCAS*-c.

It is important to mention that for the obtained *TCAS*-c and *TCAS*-ci, the multiclass cascade at the first node of the tree contains two layers, while in the case of the *TCAS*-i the first node of the tree contains only one layer. This, together with the obtained processing times and classification rates indicates that may be better to use coupled components at the first levels of the tree, while independent components should be used to train upper levels of the tree: the *TCAS*-ci has higher detection rates than the *TCAS*-c for all operation points and it is slightly slower, and the *TCAS*-ci has a better performance than the *TCAS*-i for almost the whole range (*TCAS*-i is only slightly better for false positives around 55), it is slightly faster. From the differences in the processing time and accuracy of a *TCAS*-i and a *TCAS*-c came the idea to train the *TCAS*-ci. The *TCAS*-i has a slightly better performance than the *TCAS*-c, but is much slower. We think this happens because the negatives (the "background") are "shared" by all positive classes at the first levels of the tree. Therefore it is better to shared, both features and parameters of the classifiers, as coupled components do. Later in the tree it is better to have class-specific classifiers, as independent components do, but features can still be shared.

Considering the obtained results, i.e., the *TCAS*-ci being faster than the *TCAS*-i and having an overall better performance than *TCAS*-i and *TCAS*-c, in the following analysis the *TCAS*-ci will be used.

Number of Classes

In the following we analyse the use of *TCAS* trained using different number of classes. As explained before, a n-classes *TCAS* is applied times $\frac{20}{n}$ to be able to detect faces with any rotation-in-plane. In all cases there are 20 ranges, but the number of classes considered by the classifiers is different (1, 5, 10 and 20). To cover the whole range, the classifiers are applied to rotated versions of the image if needed. The values considered for n are a 1, 5 , 10 and 20. In the case of $n = 1$, five 1-class cascades are trained, and they are applied 4 times (by rotating the image in 0, 90, 180 and 270 degrees), which gives 20 parallel cascades in total. Recall that a 1-class cascade is equivalent to a 1-class *TCAS*. We also consider the use of four 5-classes *TCAS*-ci (also by rotating the image in 0, 90, 180 and 270 degrees), two 10-classes *TCAS*-ci (by rotating the image in 0 and 180 degrees), and the use of one 20-classes *TCAS*-ci that covers the whole range (the image does not need to be rotated). In all cases the training parameters were the same, with the exception that for the 20-classes *TCAS*-ci, where 3000 negative examples per class were used instead of 3200, because of memory restrictions during training (in total 92000 training examples were used).[m]

The obtained ROC curves are presented in Figure 5.12 and Table 5.7 summarizes these results. The first thing that can be observed is that the 5-classes *TCAS*-ci and the 10-classes *TCAS*-ci have a better performance than 20 parallel cascades, and that the worst performance is obtained by the 20-classes *TCAS*-ci. The largest differences appears for low false positives, where the 5-classes and 10-classes *TCAS*s have a detection rate of up to 3 percentage points higher that the parallel cascades and from 1.5 to 7 percentage points larger than the 20-classes cascade. The 10-classes *TCAS*-ci has similar rates compared to the 5-classes *TCAS*, having a small but favorable difference of $\sim$0.5 percentage points for more than 45 false positives.

In terms of processing time, the 5-classes *TCAS*-ci (applied 4 times) is approximately 1.7 times faster than running 20 parallel cascades, the 10-classes *TCAS*-ci (applied 2 times) is 1.8 times faster

[m]Our current implementation works only on a 32bits architecture/compiler, thus is restricted to 4GB of RAM, which corresponds to $\sim$90000 training examples.

than running parallel cascades, and the 20-classes *TCAS*-ci is 1.2 times faster than running parallel cascades. The 10-classes *TCAS*-ci is 1.06 times faster than the 5-classes *TCAS*-ci, and 1.42 times faster the 20-classes *TCAS*-ci. It is hard to tell the reason of the difference in the performance of the 20-classes *TCAS*-ci with certainty, but part of its lower performance may be due to the use of a smaller training set. Nevertheless, this could be also due to some structural issue.

The training time grows close to quadratically with the number of classes, but if it is normalize by the number of training examples, 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 processing time is the number of training examples. It is important to note that the number of weak classifiers at first layer of the *TCAS*s grows logarithmically, results that was was also observed in previous experiments.

Before continuing it is important to mention that the obtained 1-class cascades, covering ranges of 18 degrees, are 5 times slower compared to the cascades obtained in Chapter 4 for a face detector. We think that this has to do with the use of a smaller training set (800 vs 5000 labeled faces). This smaller training sets seems to affect also the processing time of the *TCAS*'s (this seems to be confirmed with the results of the next section, where a multiclass detector that detects faces, but trained with a larger training set, generated a detector that is five times faster compared to the 1-class cascades obtained here).

Discussion

Out of the three variants considered to training the weak classifiers (*TCAS*-i, *TCAS*-c, and *TCAS*-ci), *TCAS*-ci presented a better performance than *TCAS*-i, being only slightly slower, and a better performance that *TCAS*-c. For applications with memory restrictions, the *TCAS*-c variant can be an interesting option, because it had a considerably smaller number of weak classifiers, and was the fastest one, however at the cost of a lower performance. The *TCAS*-ci presented a slightly better performance when compared to the use of parallel cascades and was much faster (1.7 times faster).

Table 5.6: *TCAS* weak classifier training method. Multiview RIP detection. The classifiers are trained for 5 classes (views) covering the range $[-45, 45)$.

Classifier	Training time [min]	Running time [sec]		Number of layers	Weak classifiers		Number of nodes
		640x480	320x240		total	first layer	
TCAS-i	187	7.55	1.50	61	3915	27	8
TCAS-c	161	6.44	1.30	50	2301	17	8
TCAS-ci	176	7.83	1.60	53	3913	17	8

Comparison to State of the Art Methods

Now the obtained *TCAS* (we consider the 5-classes *TCAS*-ci from the previous analysis) is compared to state of the art methods on the CMU rotated set. Figure 5.13 presents the obtained ROC curves and Table 5.8 summarizes these results.[n] The figure includes the results of the following existing methods:

[n]The numerical results for [Huang et al., 2007] and [Viola et al., 2003] were obtained using the tool GraphClick directly from the curves of presented in their papers, as the exact values were not given in the respective publications.

Table 5.7: *TCAS*-ci and number of classes. Multiview RIP detection. Each class (view) covers a range of 18 degrees. The images were rotated to 0, 90, 180, 270 degrees to cover the whole $[0,360)$ range when needed. The running time includes the time required to rotate of the images. This time, relative to the total time, is not important (it is in the order of milliseconds).
Note 1: the 20-classes *TCAS* must be evaluated once, the 10-classes *TCAS* must be evaluated twice, and the 5-classes *TCAS* (and the 5 1-class cascades) must be evaluated 4 times.
Note 2: five 1-class cascades need to be trained to cover the range $[-45,45)$.
(Sum): indicates that the values are added over the 5 cascades.

Classifier	Training time [min]	Running time [sec]		Number of layers	Weak classifiers		Number of nodes
		640x480	320x240		total	first layer	
20-classes	1862	37.84	8.05	156	10037	23	20
10-classes	461	27.27	5.76	98	6712	20	14
5-classes	176	28.89	6.10	53	3913	17	8
5 1-class cascades	5*15 = 75 (sum)	49.2	9.64	13	3251 (sum)	78 (sum)	-

- WFS-tree [Huang et al., 2007]: As previously explained, the WFS-tree consist of a tree classifier, with a similar structure to the *TCAS*, but specifically designed for the multiview detection problem. As with the *TCAS*, in [Huang et al., 2007] a multiview detector for the range $[-45,45)$ is built, and is applied four times to cover the whole range. Although the results presented in [Huang et al., 2007] can not be directly compared, some comments can be made. In terms of the results, their detection rates are from 6 to 13 percentage points higher. There are two important differences that make difficult to compare the two methods: (1) [Huang et al., 2007] makes use of granular features instead of rectangular features, – they show that these features allow to obtain higher detection rates (3 to 4 percentage points) and a faster detector –, and (2) the use of a much larger training set (30,000 labeled frontal faces vs 800).

- Two stage classification (rotation estimation, followed by de-rotation, and a neural network) [Rowley et al., 1998]: in this paper the authors proposed to de-rotate each window of the image, and then to apply a classifier. For each window, the rotation is estimated using a neural network, the windows is de-rotated, and afterwards it is classified using a second neural network. They consider several variants and show that re-training the neural network using de-rotated windows gives better results. The authors only give one operation point for each different variant, so it is difficult to compare their method to the one proposed here. They use 15720 examples generated from labeled 1048 faces as a training set. In one of their variants, their result (DR 92.4%, FP 67) is similar to the ones obtained by the *TCAS*, but their method is much slower. For each window being processed they must: estimate the rotation, de-rotate the window, and the classify it. More over their classifier is not coarse-to-fine.

- Two stage classification (rotation estimation followed by parallel cascades) [Viola et al., 2003]: Following the work of [Rowley et al., 1998], in [Viola et al., 2003] it is proposed to estimate the rotation of the window and afterwards to classify it. To do this efficiently, the authors propose to estimate the rotation using a decision tree and then to classify the window with cascade classifiers trained for specific ranges. They use 8356 labeled faces

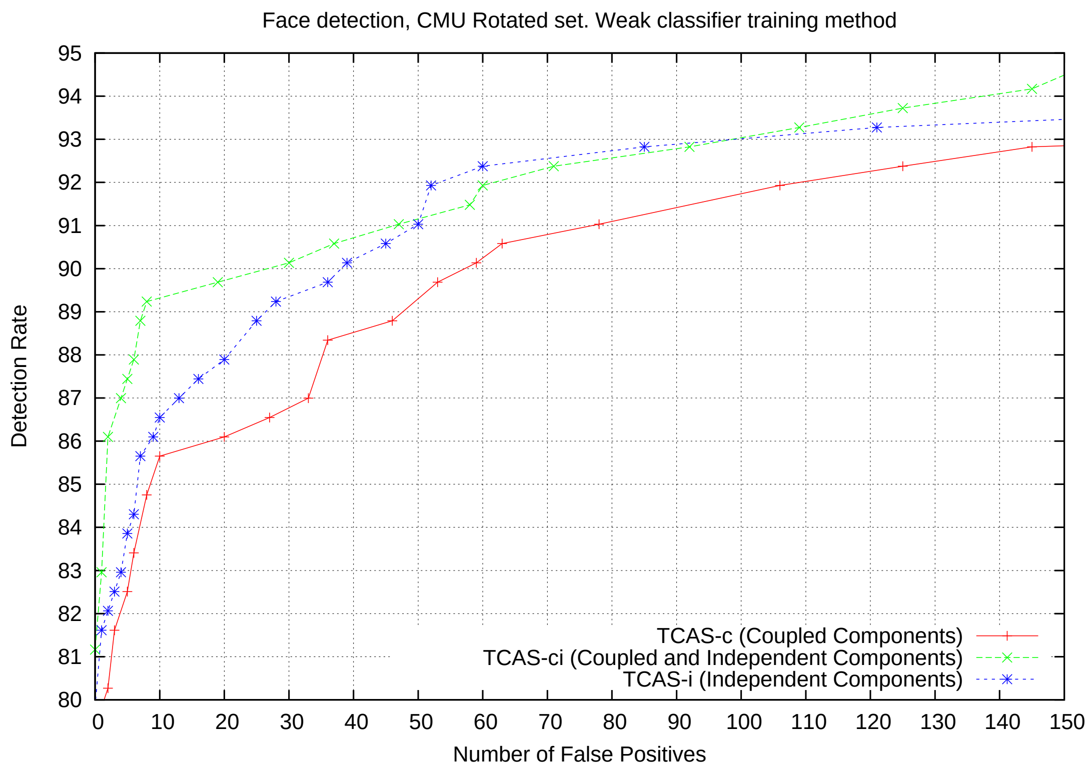


Figure 5.11: Multiview Rotation-In-Plane (RIP) Face detection ROC on the CMU Rotated set. Weak classifiers trained using Independent Components and Coupled Components: *TCAS* vs Parallel Cascades.

as training set to build each cascade, and the cascades use Haar-like wavelet (rectangular) features. Compared to the results presented here, their detection rates are 4 to 6 percentage points lower, but in terms of processing speed, their implementation is one order of magnitude faster (e.g 14ms in images of 320x240). As already mentioned, we think that difference in the processing speed with our detector is related to the size of the training set we used.

- [Li and Zhang, 2004] the authors present images with detections results on this dataset, but no ROC curves are given, therefore a comparison is not possible.

Figures 5.14 to 5.16 show some results on images from the CMU Rotated Set, and Figures 5.17 to 5.20 show results on other images containing several rotated faces. All of these results were obtained using the same operation point of the 5-class *TCAS*-i detector.

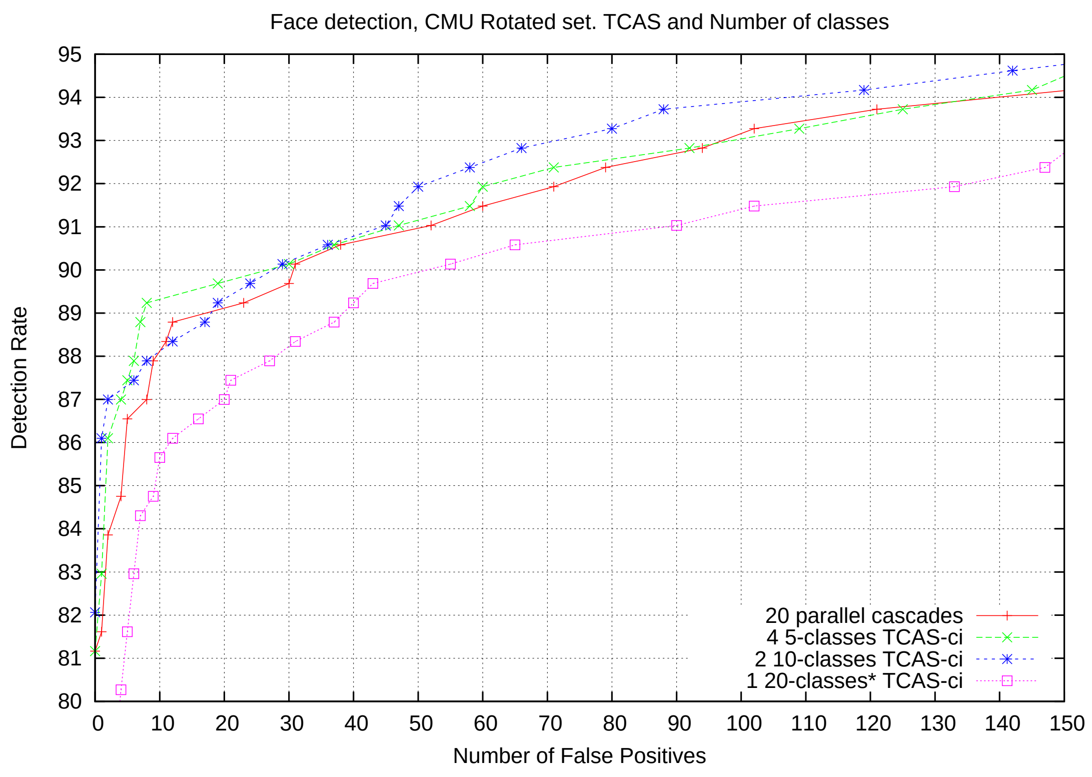


Figure 5.12: Multiview Rotation-In-Plane (RIP) Face detection ROC on the CMU Rotated set. Number of classes in a *TCAS*-ci.
* Note: The 20-classes case was trained with fewer examples per class. See text for details.

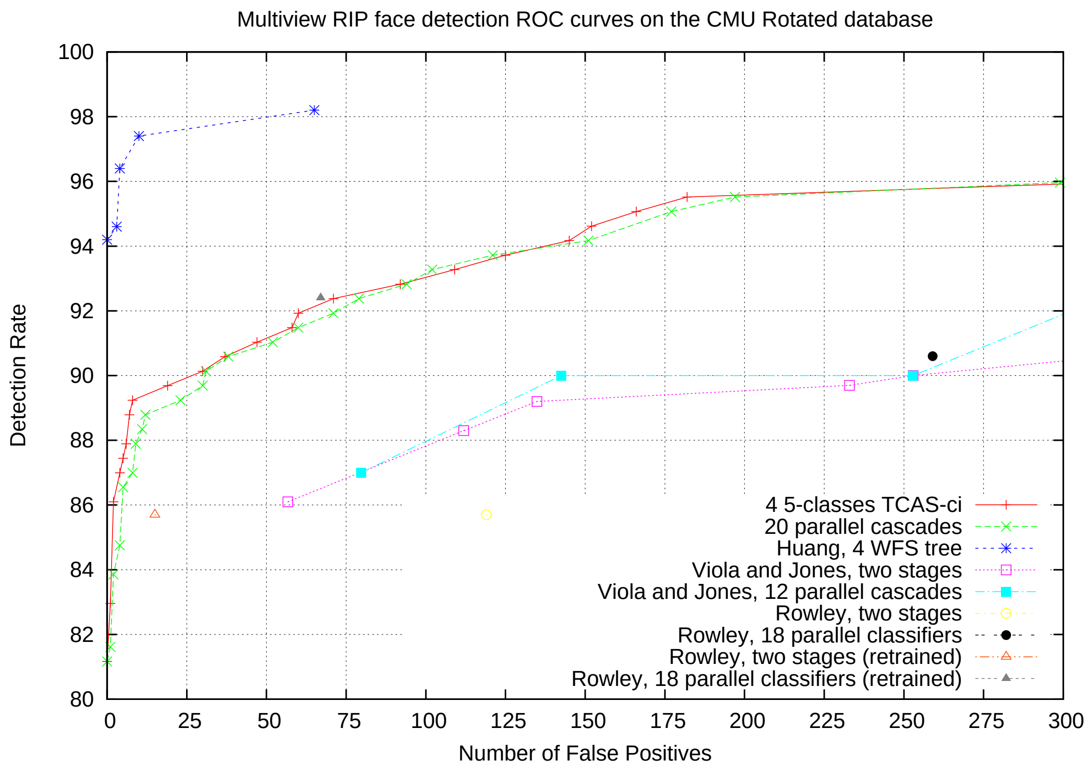
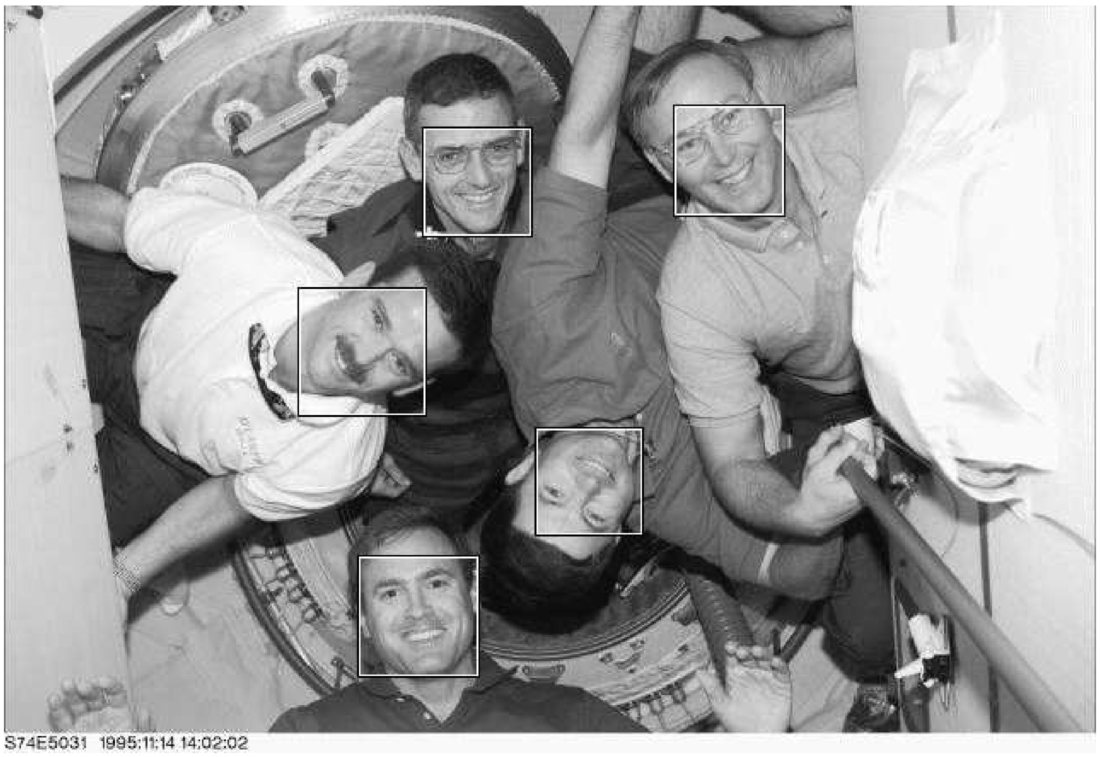


Figure 5.13: Rotation-In-Plane (RIP) Face detection ROC on the CMU Rotated set. Comparison with state of the art methods.

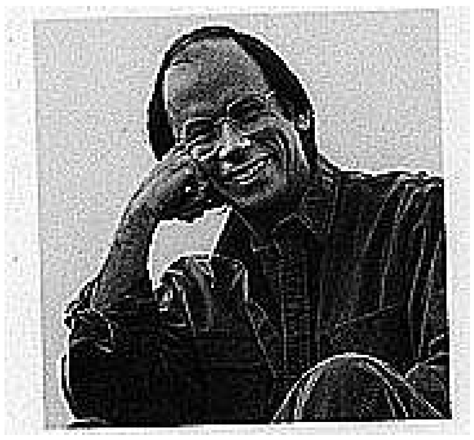
Table 5.8: Comparative evaluation of the face detector on the CMU Rotated Set.
* The numerical results for [Huang et al., 2007] and [Viola et al., 2003] were obtained using the tool GraphClick directly from the curves of presented in their papers, as the exact values were not given in the respective publications.
$^+$: case where the authors re-trained their classifiers using de-rotated windows.

False Positives	0	4	8	10	15	23	38	57	60
[Rowley et al., 1998] two stages									
[Rowley et al., 1998] parallel									
[Rowley et al., 1998] two stages $^+$					85.7				
[Rowley et al., 1998] parallel $^+$									
[Huang et al., 2007] WSF-tree *	94.2	96.4		97.4					
[Viola et al., 2003] two stage *								86.1	
[Viola et al., 2003] parallel *									
TCAS-ci	81.2	87.9	89.2				90.6		91.9
parallel cascades	81.2	84.8	87.0			89.2	90.6		91.5

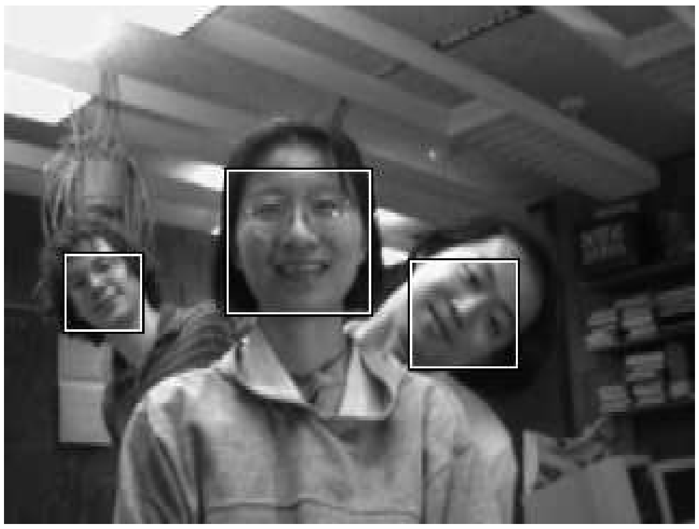
False Positives	65	67	80	112	119	135	143	233	259
[Rowley et al., 1998] two stages					85.7				
[Rowley et al., 1998] parallel									90.6
[Rowley et al., 1998] two stages $^+$									
[Rowley et al., 1998] parallel $^+$		92.4							
[Huang et al., 2007] WSF-tree *	98.2								
[Viola et al., 2003] two stage *				88.3		89.2		89.7	
[Viola et al., 2003] parallel *			87.0				90.0		
TCAS-ci							94.2		
parallel cascades			92.4						



(a) Detected faces: 5 out of 5; False positives: 0. Image size: 640x438.

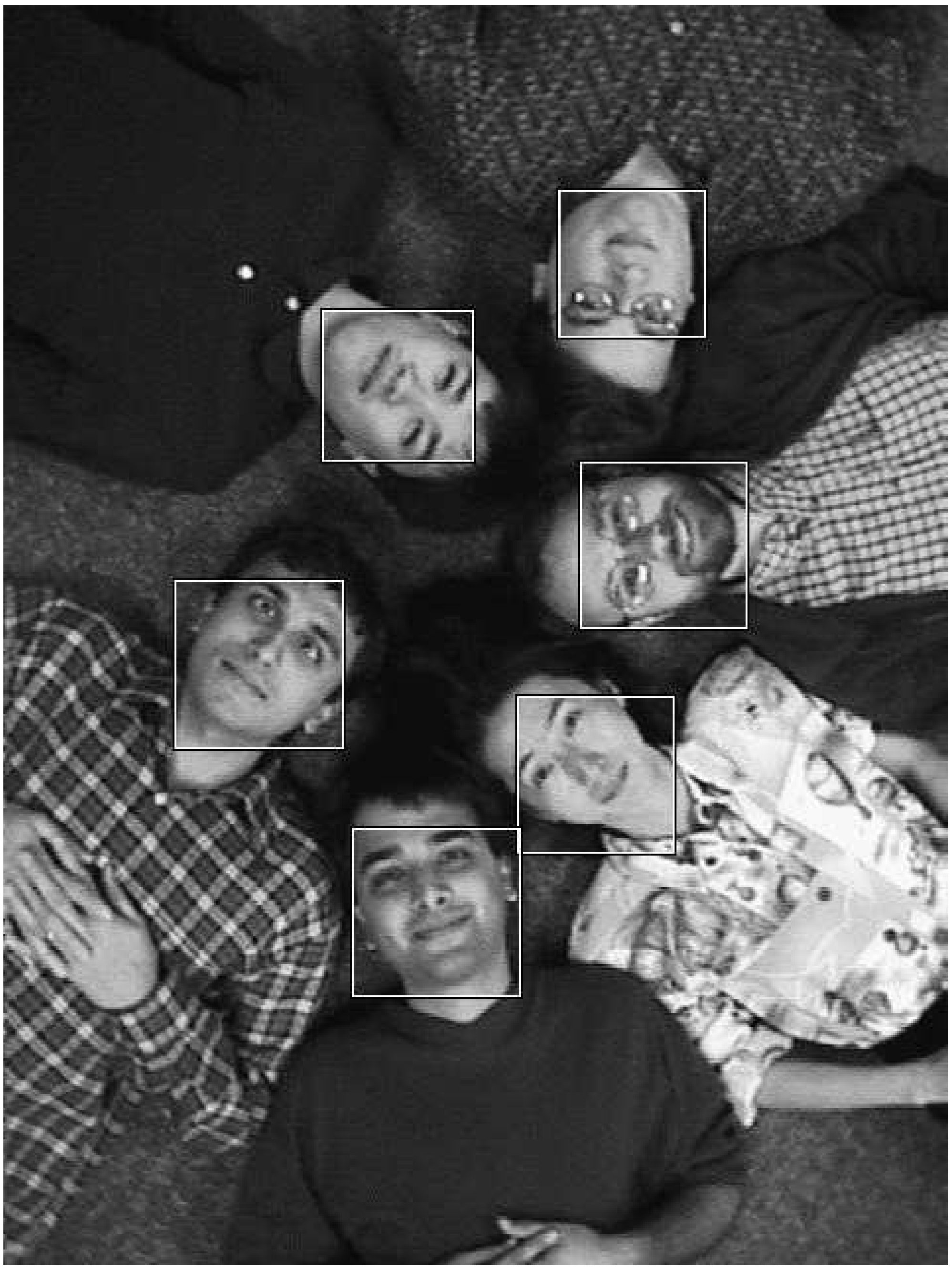


(b) Detected faces: 0 out of 0; False positives: 0. Image size: 180x166



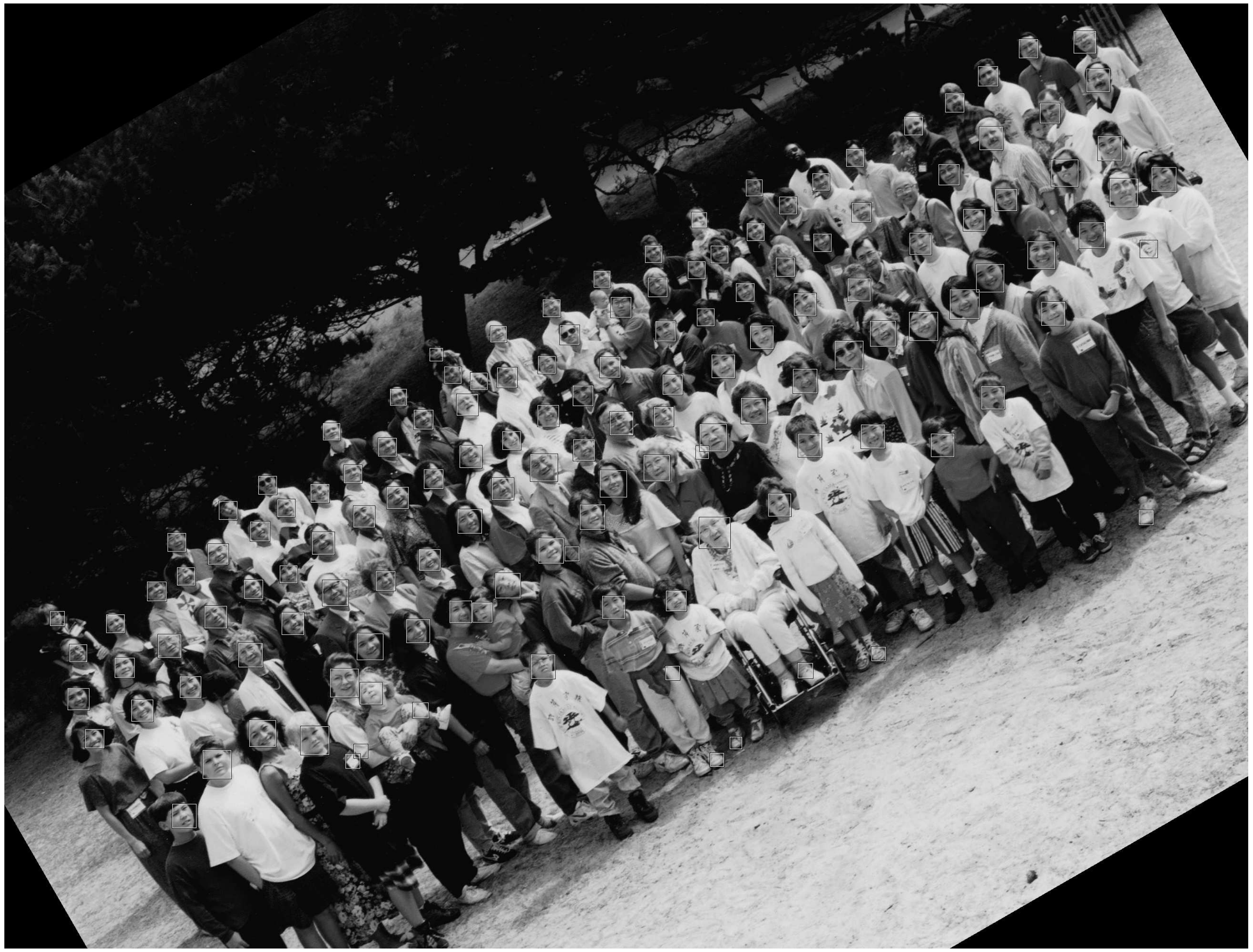
(c) Detected faces: 3 out of 3; False positives: 0. Image size: 320x240

Figure 5.14: Example results of the RIP face detector. Images from the CMU Rotated Set.



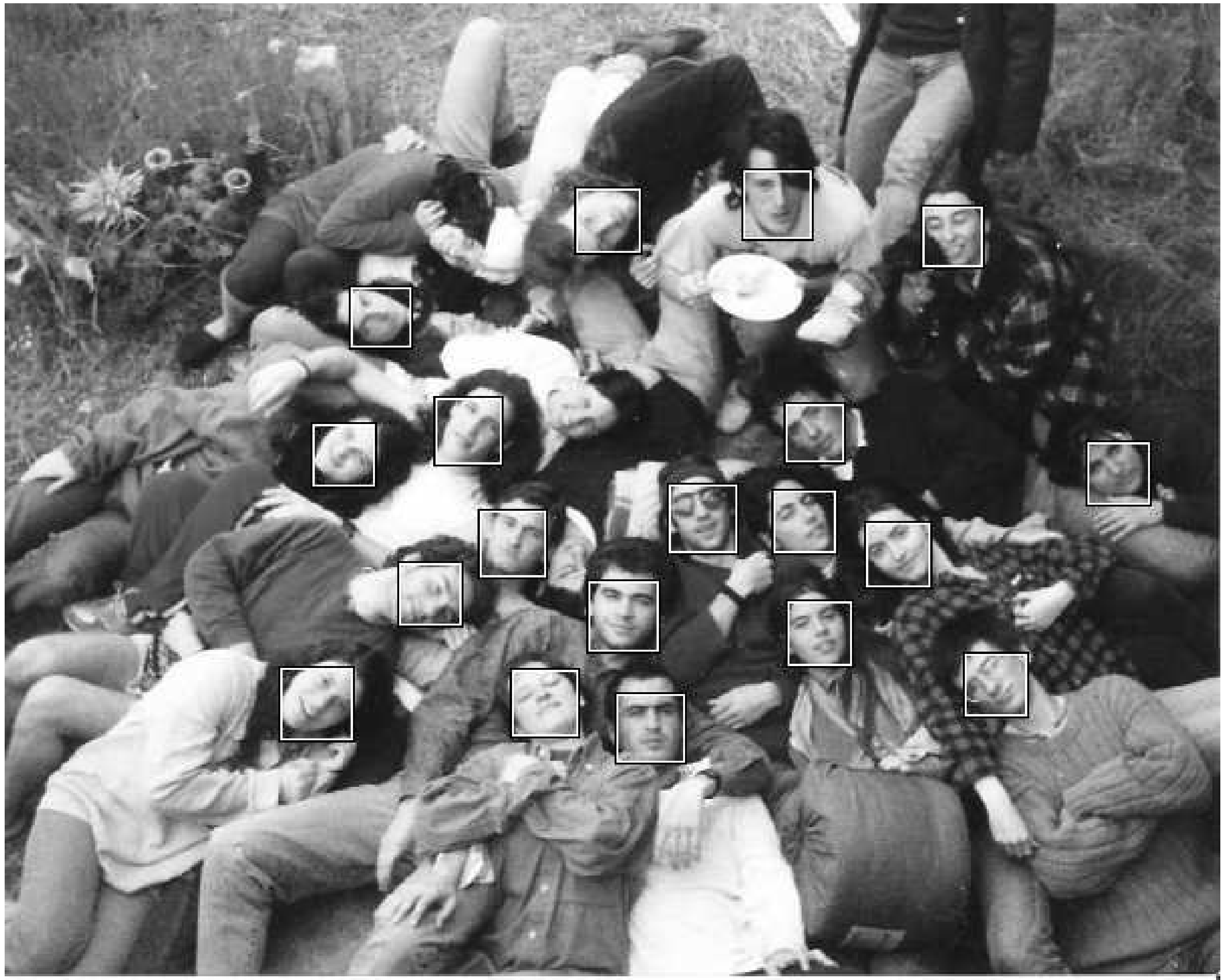
(a) Detected faces: 6 out of 6; False positives: 0. Image size: 480x640.

Figure 5.15: Example results of the RIP face detector. Image from the CMU Rotated Set.



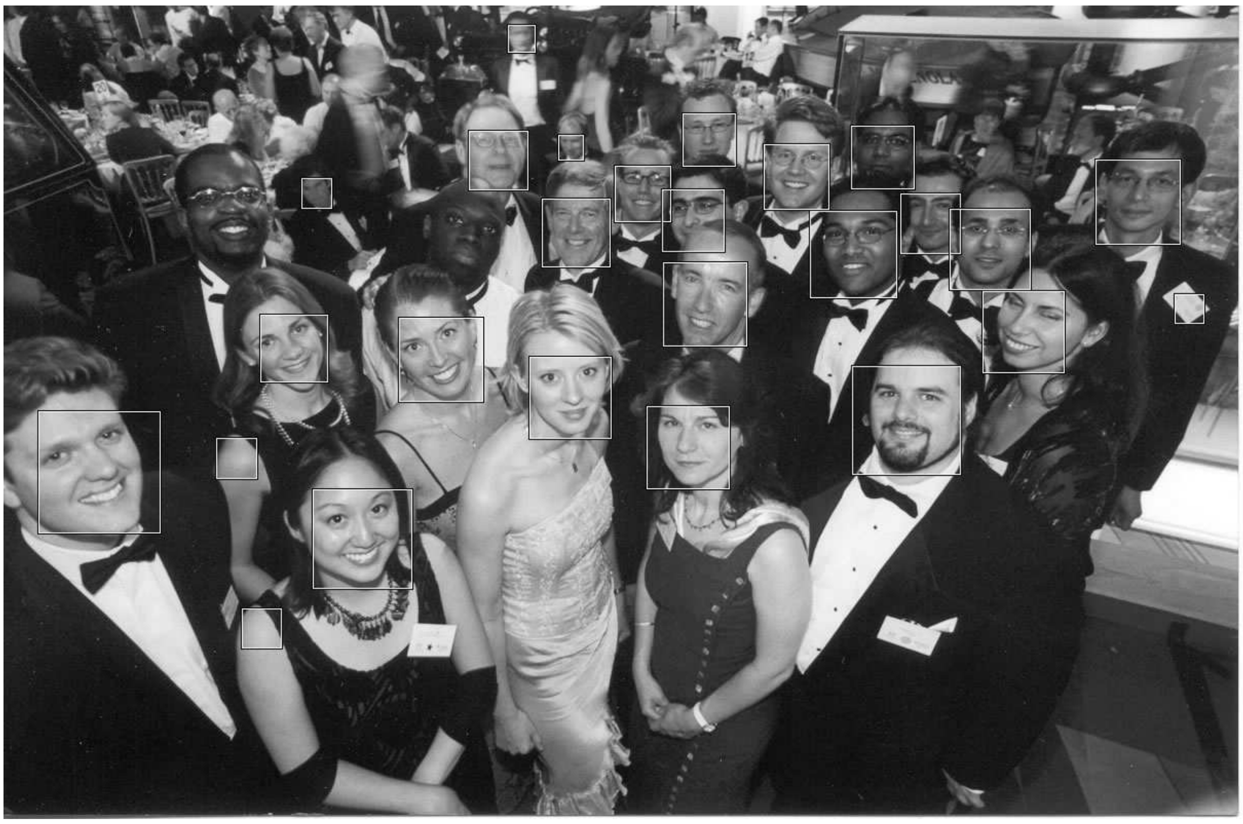
(a) Detected faces: 127 out of 134; False positives: 14. Image size: 2615x1986.

Figure 5.16: Example results of the RIP face detector. Image from the CMU Rotated Set

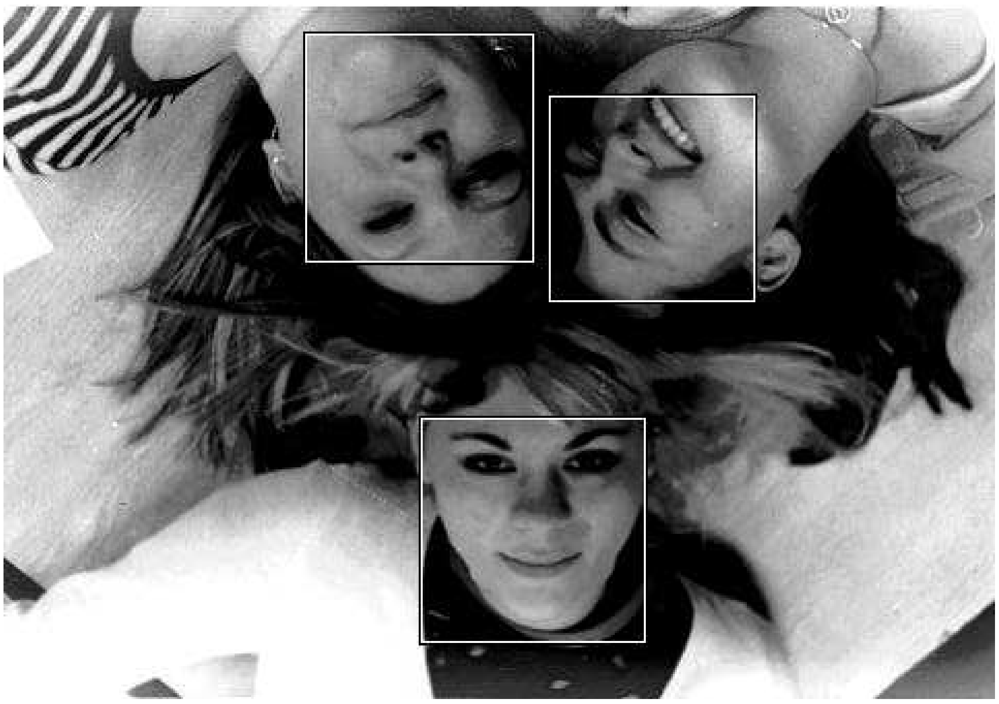


(a) Detected faces: 19 out of 22; False positives: 0. Image size: 604x497

Figure 5.17: Example results of the RIP face detector.

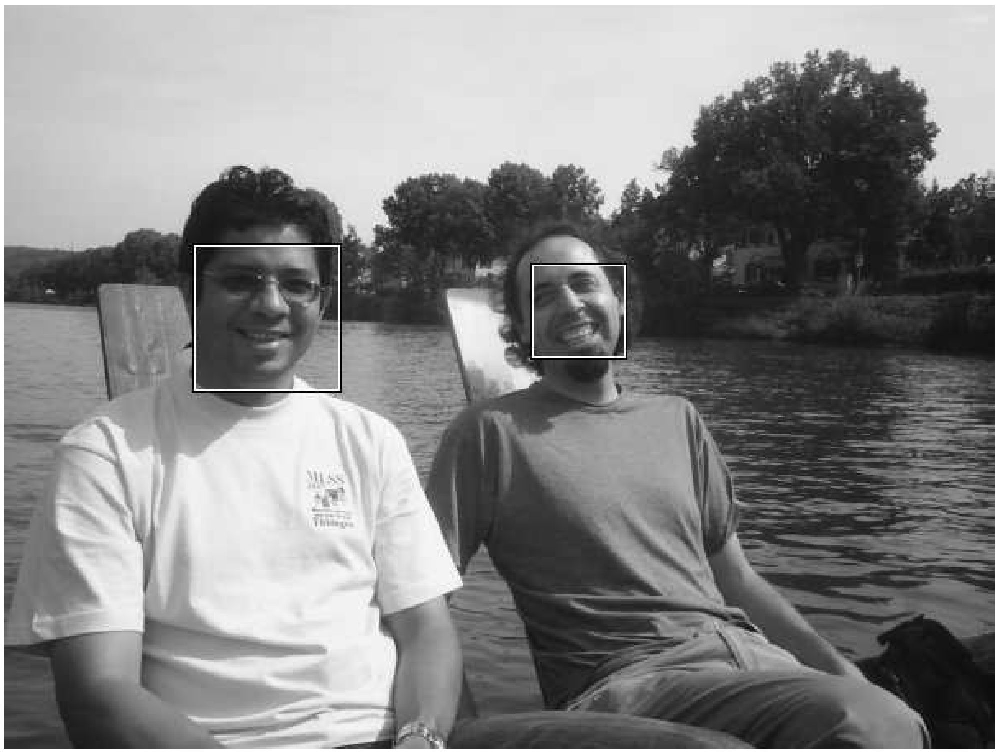


(a) Detected faces: 24 out of 27; False positives: 3. Image size: 1209x792



(b) Detected faces: 3 out of 3; False positives: 0. Image size: 336x480

Figure 5.18: Examples results of the RIP face detector.


(a) Detected faces: 2 out of 2; False positives: 0. Image size: 604x453

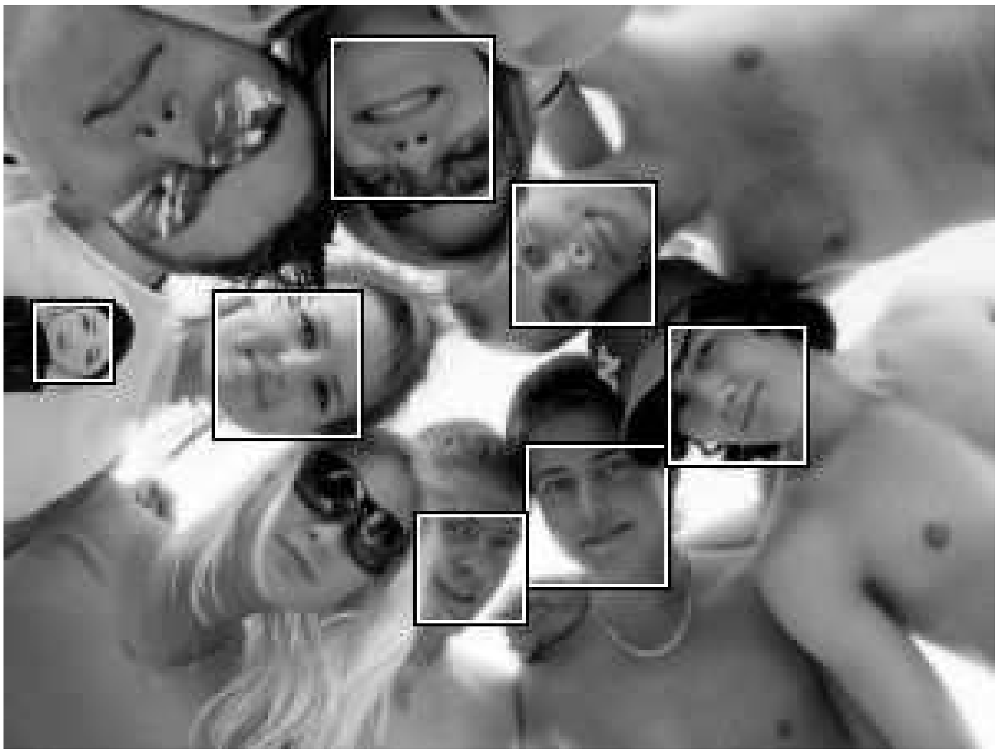

(b) Detected faces: 7 out of 9; False positives: 0. Image size: 300x225

Figure 5.19: Examples results of the RIP face detector.

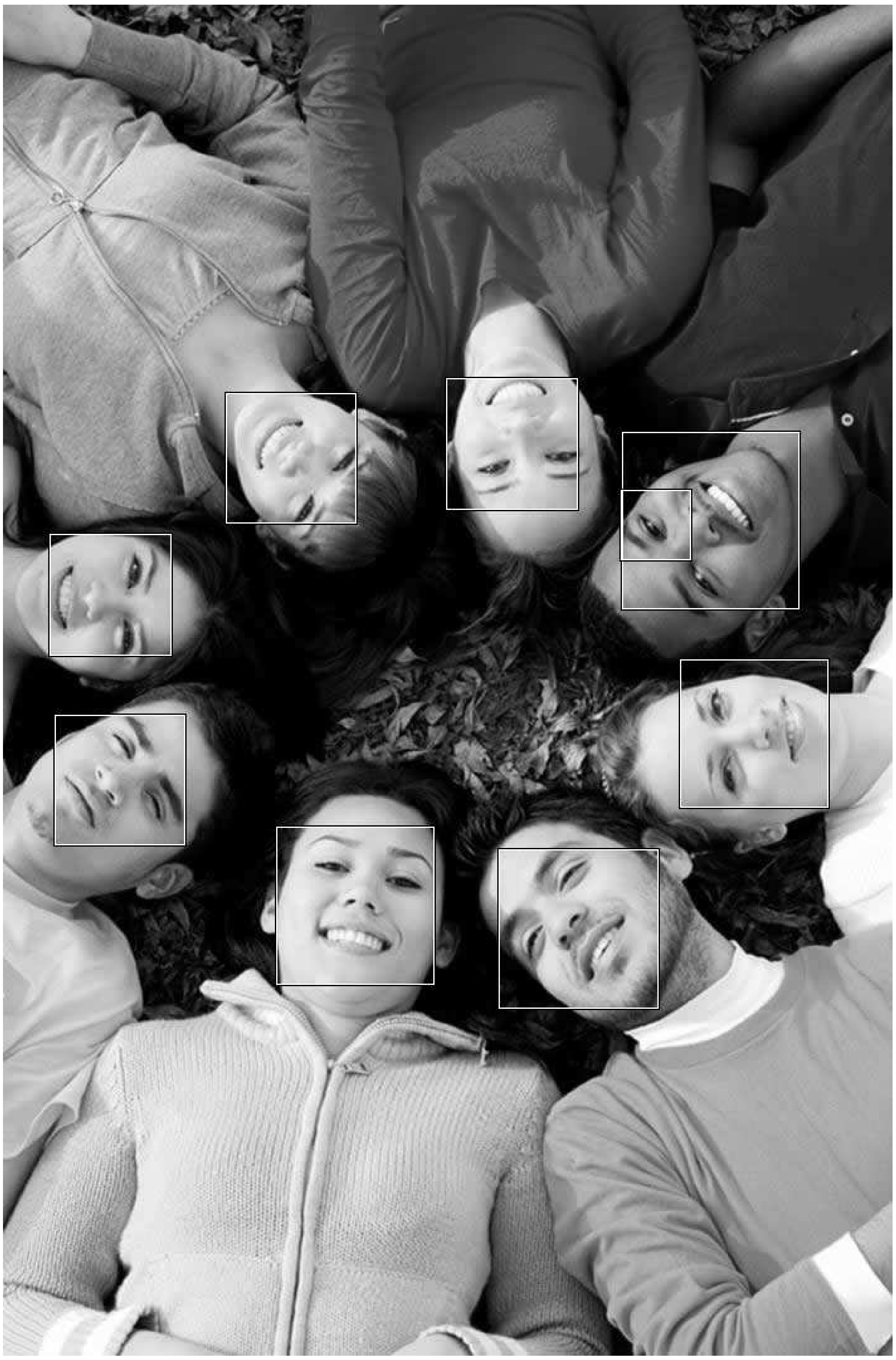

(a) Detected faces: 8 out of 8; False positives: 1. Image size: 707x1075

Figure 5.20: Examples results of the RIP face detector.

5.6.2 Hand and Face Detection

Now we evaluate the proposed framework and classifiers on a multiclass setting: the problem of hand and face detection. More specifically, three classes are considered: frontal faces, frontal left fists and frontal right fists. With "frontal" we refer to objects presenting low in-plane rotations (up to ± 15 degrees). See Figure 5.25 (a) for some examples of what we considered to be a frontal right hand fist, a frontal left hand, and a frontal face.

The evaluation consist of building several cascades and *TCAS*s classifiers for different combinations of classes and number of classes. Afterwards the processing time of the obtained classifiers was measured, as well as the accuracy of the fastest ones. Given that there is no standard datasets for this problem, we present ROC curves for frontal faces on a standard database. For the detections of hands, we present images with detection results.

The main goal of this experiment was to evaluate the performance of using a *TCAS* on a multiclass setting and compare it to the use of a one-class cascade trained for a specific class using the same training sets and parameters.

First we considered all possible combinations of 1-class, 2-classes, and 3-classes object detectors built using cascades and *TCAS* classifiers trained using weak classifier with independent components. Afterwards we analyse the use of nodes with two siblings (split2) and three siblings (split3). Finally we evaluate the use of coupled and independent components in the training of the weak classifiers.

See Table 5.10 (first and second columns) for a summary of the built detectors. We trained one 1-class cascade for faces and one for left hands. It was not possible to obtain multiclass cascades (two-classes and three-classes), because the training procedure did not converge (adding weak classifiers did not improved the classification rates and desired *fpr* and *tpr* per class and per layer were not achieved). On the contrary the *TCAS* classifiers were built for two-classes and three-classes problems.

The classifiers were built using 4900 face examples, 4900 left hand fist examples and 4900 right hand fist examples not appearing in any of the images used in the evaluation. In all experiments, all used parameters and (initial) training sets were exactly the same. Rotated and non-rotated rectangular features were used in the first two layers, while in later layers mLBP features were considered. The *fpr* per layer was set to 0.35, the *tpr* per layer per class was set to 0.999, and the target *fpr* was 10^{-6}, which is obtained with cascades of 13 layers.

The presented ROC curves are obtained for the BioID database and correspond results for the problem of frontal face detection. We use the BioID database (1,521 images and 1521 faces) because it does not contain any hand.

Processing Times

Table 5.10 presents the processing times of the built object detectors, and Table 5.9 presents a summary of the images used for this evaluation. These images are shown in Figures 5.23, 5.24 and 5.25. The presented detection results were obtained using the 3-classes *TCAS*-i detector (split3). The CMU-MIT database was used to evaluate the processing time in these experiments, because it presents a larger variability in the content of the images compared to other similar databases (e.g BioID).

The first thing that can be noticed from Table 5.10 is that in all cases (2 and 3 classes) running a *TCAS*-i is more efficient than: (1) running several parallel 1-class cascades, and (2) running in parallel a 1-class cascades and *TCAS*-i for the remaining classes.

In the two-classes problem (left and right fist hands; and face and left hand), the 2-classes *TCAS*-i is faster than running two cascades in parallel. The average processing time on the CMU-MIT database is 1.36 times faster than the parallel cascades in the case of hand and face detection, and 1.72 times faster in the left hand and right hand detection problem.

In the three-classes problem when building the trees, the first nodes of the tree were trained considering two siblings (split2) and three siblings (split3). Both trainings converged, but using three siblings produced a slight faster classifier. The most important result is that when any three-classes *TCAS* is compared to the use of parallel cascades, the gain is close to 1.6 times on the CMU-MIT database, and that for some particular images the gain is more than two-fold (Judybats, Addam, and HF34; see Figures 5.23 (a)-(b) and 5.24 (b)). For the remaining images, which contain complex backgrounds and many objects (HF75 and HF51; see Figures 5.24 (a) and 5.25 (a)) the gain is about 1.3 to 1.4 times.

Note that all detectors are particularly slow on the images HF51 and HF75 (Figures5.24 (a) and 5.25 (a)), even if the objects of interest (faces and hands) are removed (HR75r, Figure 5.24 (a)). We think this is due to the strong horizontal structure of the background of that images (e.g. black strips), which resemble parts of the faces and the hand fists gestures. For example if the images HF34 and HF51 are compared, images of the same size and taken with the same camera, the ratio between their processing times goes from 2.4 to 3.4 for different detectors, having the largest gain when a *TCAS* is used. This shows that the use of a *TCAS* is even more helpful in difficult cases.

TCAS Variants

As shown in Table 5.10, in the three-classes case, when comparing the use of independent components (*TCAS*-i), and coupled components (*TCAS*-c) there are no major differences between them in terms of processing time, with the variant *TCAS*-i (split3) being just slightly faster, followed by *TCAS*-i (split3) and *TCAS*-c (split3). Note that this is a three-classes problem, thus when three siblings are used (split3) *TCAS*-c is equivalent to *TCAS*-ci.

In order to evaluate the accuracy of the *TCAS*, we consider the problem of frontal face detection. The goal of this analysis is to compare the performance of the multiclass classifier, when used to detect only one particular class.

Figure 5.21 (a) presents ROCs of the 3-classes *TCAS*-i detectors when using 2 siblings (split2) and 3 sibligns (split3) per node. As it can be observed, the *TCAS* trained using three siblings per node presents a slightly better accuracy than the one trained having two siblings, with detection rates between 0.2 and 1 percentage points higher for false positive rates larger than 10. For lower false positive rates, the difference becomes much larger, with a difference over 4 percentage points for 0 false positives.

In Figure 5.21 (b) we compare the use of the different variants to train the weak classifiers when three siblings per node are used (split3). As it can be observed, the use of coupled components (*TCAS*-c) gives the better results than the use of independent components (*TCAS*-i). The differences in detection rate between the *TCAS*-c and the *TCAS*-i is close to 1.5 percentage points for low false positive points (e.g. 5 FP), and close to 0.5 for 50 false positives. Recall that the obtained *TCAS*-i was only 1.08 times faster that *TCAS*-c.

In Figure 5.22 the best *TCAS* (*TCAS*-c) is compared to the use of a one-class cascade trained specifically for that class. 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 low false positives (5 FP). This same *TCAS*-c classifier (see Table 5.10) is 1.56 times faster than the use of parallel cascades in the CMU-MIT database. Note that this cascade and this *TCAS* have accuracies and processing times of the

Table 5.9: Images used for testing the face and hand detection problem. The CMU-MIT face database consist of 130 images and 507 faces.

Image/DB Name	Size	DB	Faces		Hand Fists		
			frontal	profile	left	right	other
Judybats	716x684	CMU-MIT	5	0	0	0	0
Addam’s	864x890	CMU-MIT	7	0	0	0	1
FH75	800x600	Personal	3	2	3	3	4
FH75r	800x600	Personal	0	0	0	0	0
FH34	800x600	Personal	0	0	0	0	0
FH51	800x600	Personal	4	1	5	5	0
CMU-MIT	432x437 (avg)	CMU-MIT	507	0	-	-	-

order of magnitude of the ones presented in Chapter 4.

Table 5.10: Processing times [sec] of frontal hand (Right and Left fist) and frontal face detector(s) on different images. See text for comments and Table 5.9 for details on the images. Best results for each number of classes are shown in **bold**.
* Note: All detectors are particularly slow on HF51 and HF75, even if faces and hands are removed (HR75r).

Classifier	Classes	Judybats	Addam	HF75(r)*	HF34	HF51	CMU-MIT
One-class							
1-class cascade	Face	1.31	1.76	1.41(1.30)	0.85	1.48	0.66
1-class cascade	Left Hand	0.97	1.13	1.74(1.56)	0.65	1.85	0.52
Two-classes (Left and Right Hands)							
Two 1-class cascades	Left & Right Hand	1.94	2.26	3.48(3.12)	1.3	3.7	1.05
2-classes cascade	Left & Right Hand	the training did not converge					
2-classes *TCAS*-i	Left & Right Hand	**1.05**	**1.30**	**2.03(1.87)**	**0.80**	**2.19**	**0.61**
Two-classes (Left Hand and Frontal Face)							
Two 1-class cascades	Left Hand & Face	2.28	2.89	3.15(2.86)	1.5	3.33	1.18
2-classes cascade	Left Hand & Face	the training did not converge					
2-classes *TCAS*-i	Left Hand & Face	**1.57**	**1.79**	**2.33(2.15)**	**0.78**	**2.47**	**0.87**
Three-classes (Frontal Face, Left Hand and Right Hand)							
Three 1-class cascades	Face & Hands	3.25	4.02	4.89(4.42)	2.15	5.18	1.70
1-class + 2-classes *TCAS*	Face & (L&R Hand)	2.36	3.06	3.44(3.17)	1.65	3.67	1.27
1-class + 2-classes *TCAS*	L & (R Hand & Face)	2.53	2.92	4.07(3.43)	1.43	4.32	1.39
3-classes cascade	Face & Hands	the training did not converge					
3-classes *TCAS*-i (split2)	Face & Hands	1.80	2.01	**2.96(2.80)**	**0.99**	**3.14**	1.03
3-classes *TCAS*-i (split3)	Face & Hands	**1.70**	**1.99**	3.18(2.90)	**1.00**	3.44	**1.01**
3-classes *TCAS*-c (split3)	Face & Hands	1.84	2.05	3.39(3.14)	1.05	3.69	1.09

Discussion

Under this multiclass setting, the obtained *TCAS*-c classifier (equivalent to a *TCAS*-ci in this case) was 1.56 times faster than running cascades in parallel, and had up to 1 percentage points higher detection rates when detecting faces. This is an important result, showing that the proposed *TCAS* classsifier can not only improve the processing speed of a multiclass detection system, but also it accuracy, when compared to the use of parallel cascades. As in the RIP face detection problem, the best performing *TCAS* was obtained using coupled components (*TCAS*-c), result indicating that coupled components should be used at the first layers of the classifiers.

5.7 Conclusions

In the present chapter methods for building multiclass object detectors were proposed. The proposed classifier is called *TCAS* and corresponds to a multiclass classifier with a nested tree structure. In order to build this classifier, we introduce coarse-to-fine (CTF) multiclass nested cascades and analyse the use of three variants of multiclass weak classifiers, including one (*coupled* components) proposed here. We also extend the cascade classifier and training algorithms presented in Chapter 4 in order to train the multiclass nested cascades.

We evaluated the system by building multiview face detection systems, considering different numbers of views. *Coupled* components and two other variants were used to build multiclass CTF nested cascade classifiers. Good performance and reasonable processing times were obtained when coupled components and independent components were used. Also, compared to the use of joint components, the training time was reasonable low and it did not grow exponentially. The use of CTF search in the object target space considerable improved the accuracy and the processing speed of the multiclass cascade.

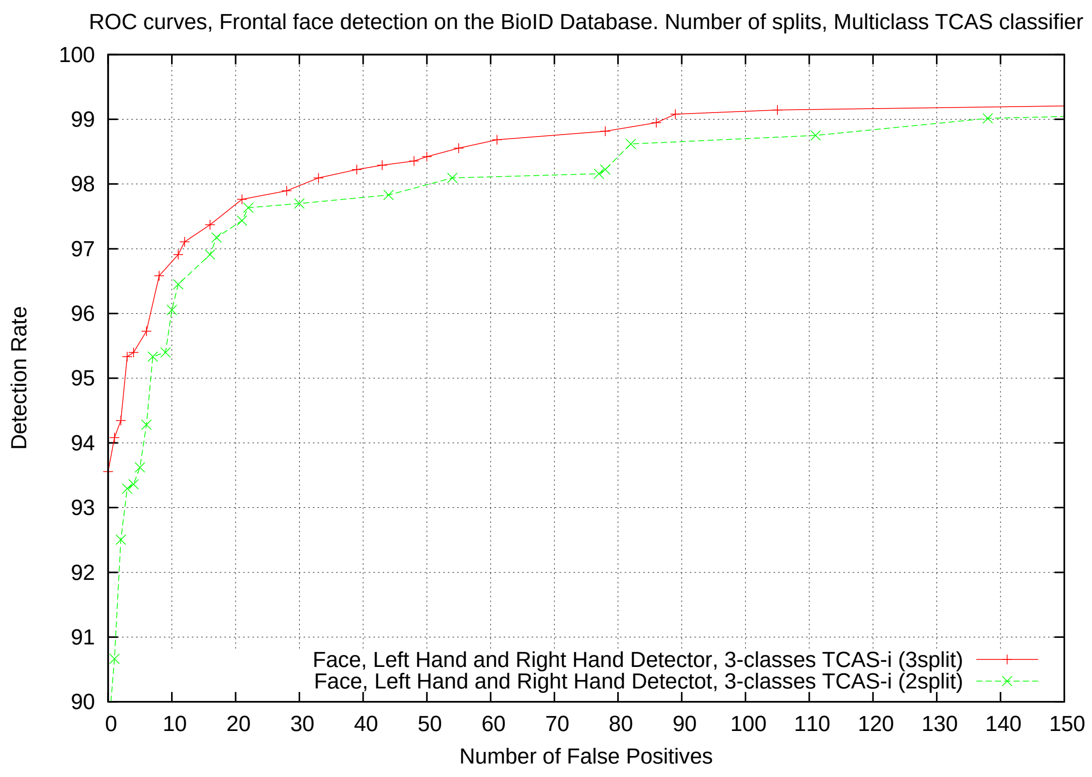
Using the *TCAS* structure we were able to build RIP multiview face detectors. Three variants were considered: training the weak classifiers using (a) independent components (*TCAS*-i), (b) coupled components (*TCAS*-c), and (c) training the first layers using coupled components, and using independent components afterwards (*TCAS*-ci). *TCAS*-ci presented a better performance than *TCAS*-i, being only slightly slower. The variant *TCAS*-c had a lower performance than *TCAS*-ci, but faster, and had a considerably smaller number of weak classifiers, which can be of use in applications with memory restrictions. The *TCAS*-ci was slightly better when compared to the use of parallel cascades, and 1.7 times faster. In addition, using this *TCAS*, the second best published results on standard database were obtained, results obtained using a training set one to two orders of magnitude smaller than existing methods. On the contrary, the running time compared to similar methods is close to one order of magnitude higher, which is likely to be due to the use of a small training set.

The *TCAS* classifier was also evaluated on a multiclass setting, by building multiclass detectors of frontal faces, frontal right fist and frontal left fist. In terms of processing time, the obtained *TCAS*-c classifier built for the three classes – which is equivalent to a *TCAS*-ci in this case – was 1.56 times faster than running cascades in parallel, and had up to 1 percentage points higher detection rates in the problem of face detection compared to a 1-class cascade trained using the same data. The best performing *TCAS* was obtained using coupled components (*TCAS*-c), which in this case gave better results that using independent components (*TCAS*-i), but being slight slower. In this problem, as in the RIP face detection problem, the use of coupled components in the first layers and independent components in the remaining ones gave the best results.

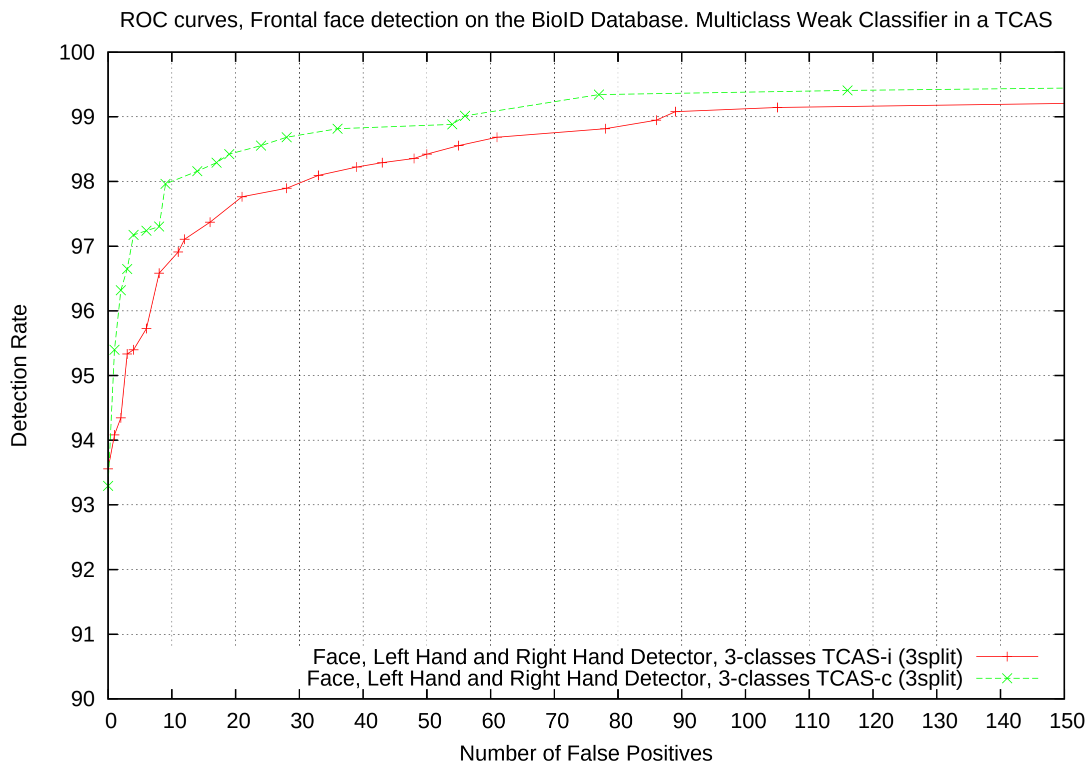
Regarding the training time, the proposed methods are efficient, with training times of only hours for problems with a few classes (3-5) and mid-size training sets (5000), which is quite fast compared to existing methods that require days to train similar systems. These training times are obtained thanks to the use of simple and effective procedures, with methods that are linear on the number of training examples and at most quadratic in the number of classes.

Possible future research directions include: (1) combining this work with the ideas of [Wu and Nevatia, 2007] to build jointly multiclass and multiview/pose systems, (2) to automatically group the classes instead of using predefined groups [o], and (3) the evaluation of the system using other features, and on other detection problems such as car detection and pedestrian detection.

[o]Preliminary analysis with methods not presented in this document shows some promising results.



(a) Training of the multiclass *TCAS*-i using three (3split) and in two (2split) siblings.



(b) Training of multiclass *TCAS* using different kinds of weak classifiers: coupled components (*TCAS*-c), and independent components (*TCAS*-i) .

Figure 5.21: Face detection ROC curves on the BioID set. The *TCAS*s were trained to detect Frontal Faces, Left Fist and Right Fist Hands, but are used only to detected faces.

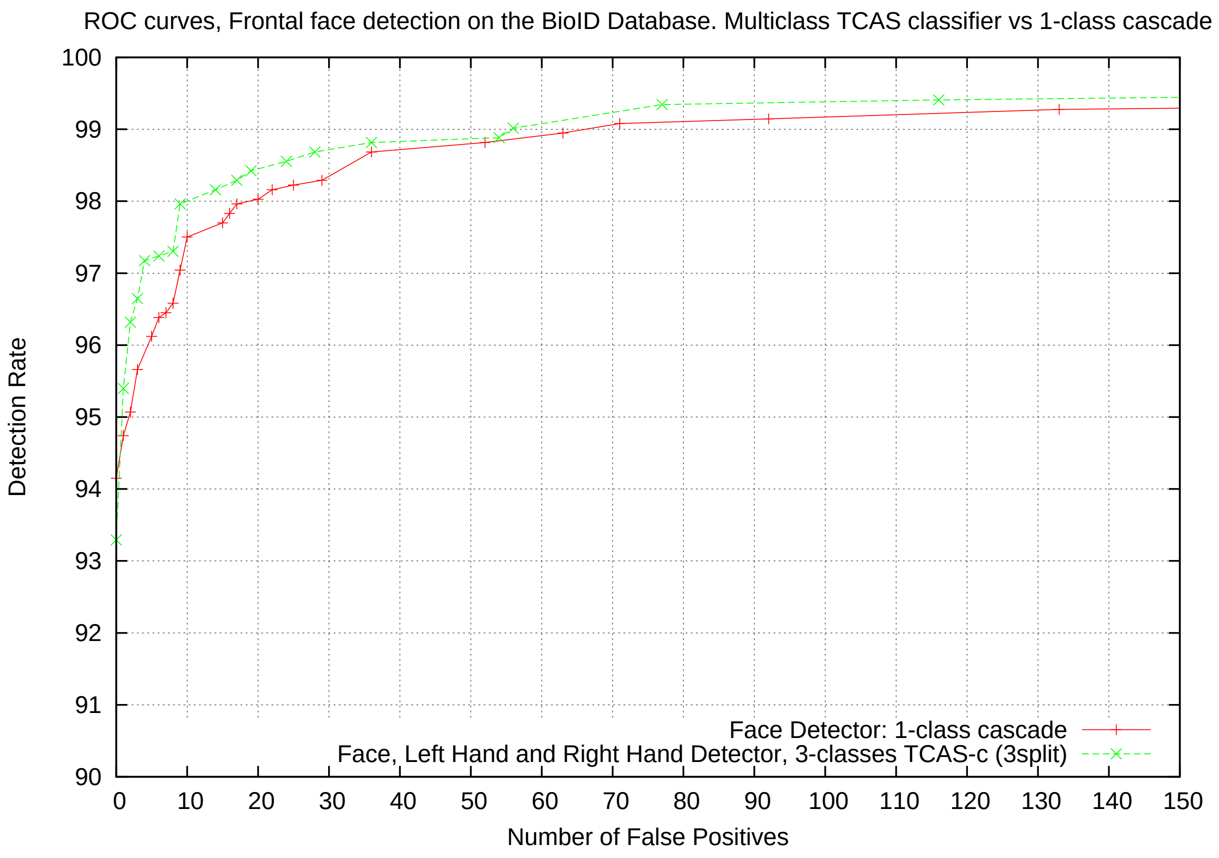
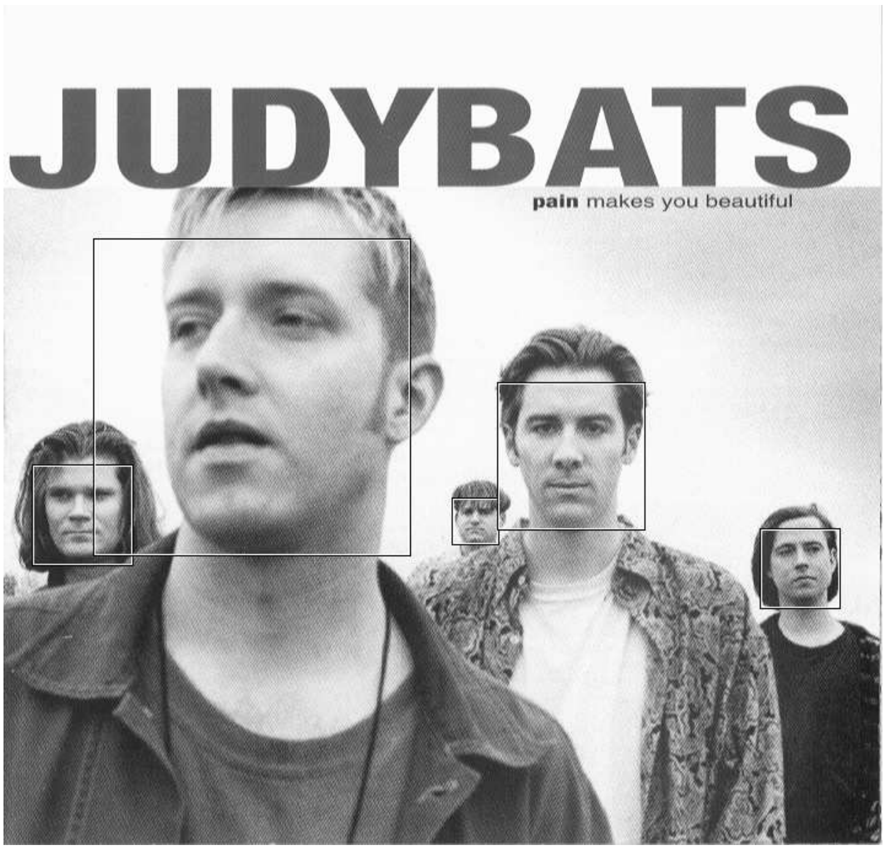
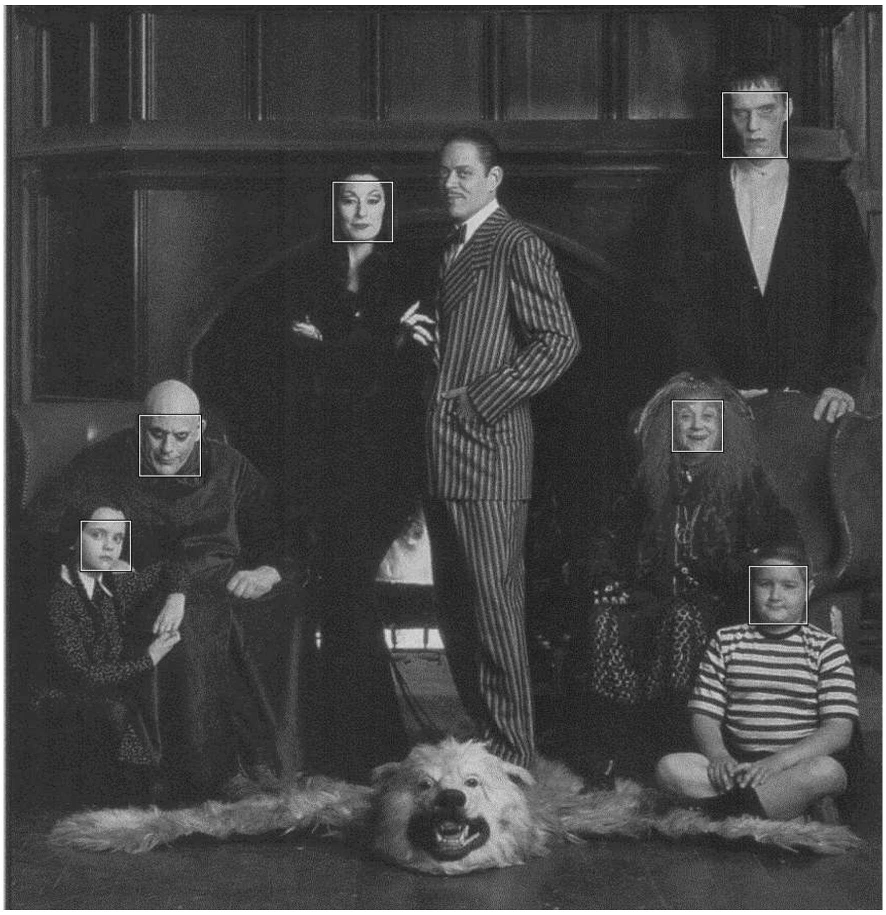


Figure 5.22: Face detection ROC curves on the BioID set. Comparison of using a multiclass (3 classes) *TCAS*-c with a 1-class cascade on the face detection problem. The *TCAS* was trained to detect Frontal Faces, Frontal Left Fist Hands and Frontal Right Fist Hands, but here is used only to detected faces.

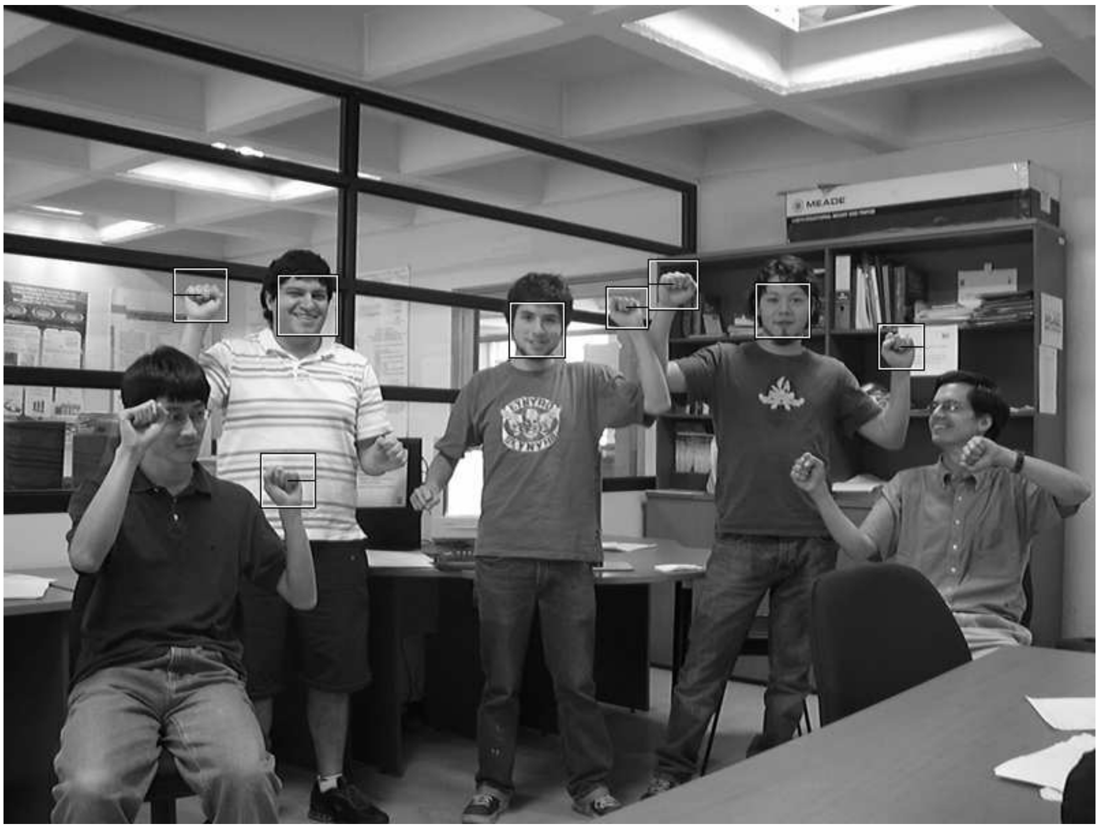


(a) Judybats

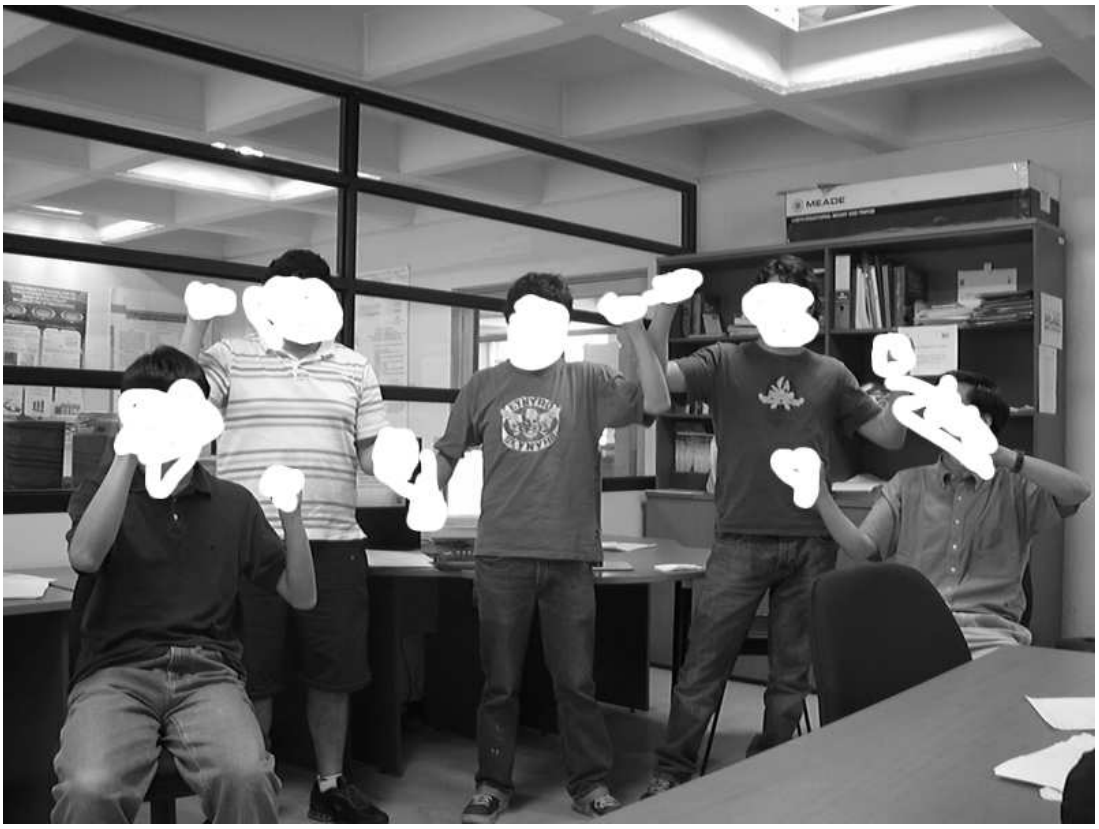


(b) Addam's

Figure 5.23: Hand and Face detection results. Problem: 3-classes (frontal left fist, frontal right fist and frontal faces). In (a) all faces are detected. In (b) one face is not detected.

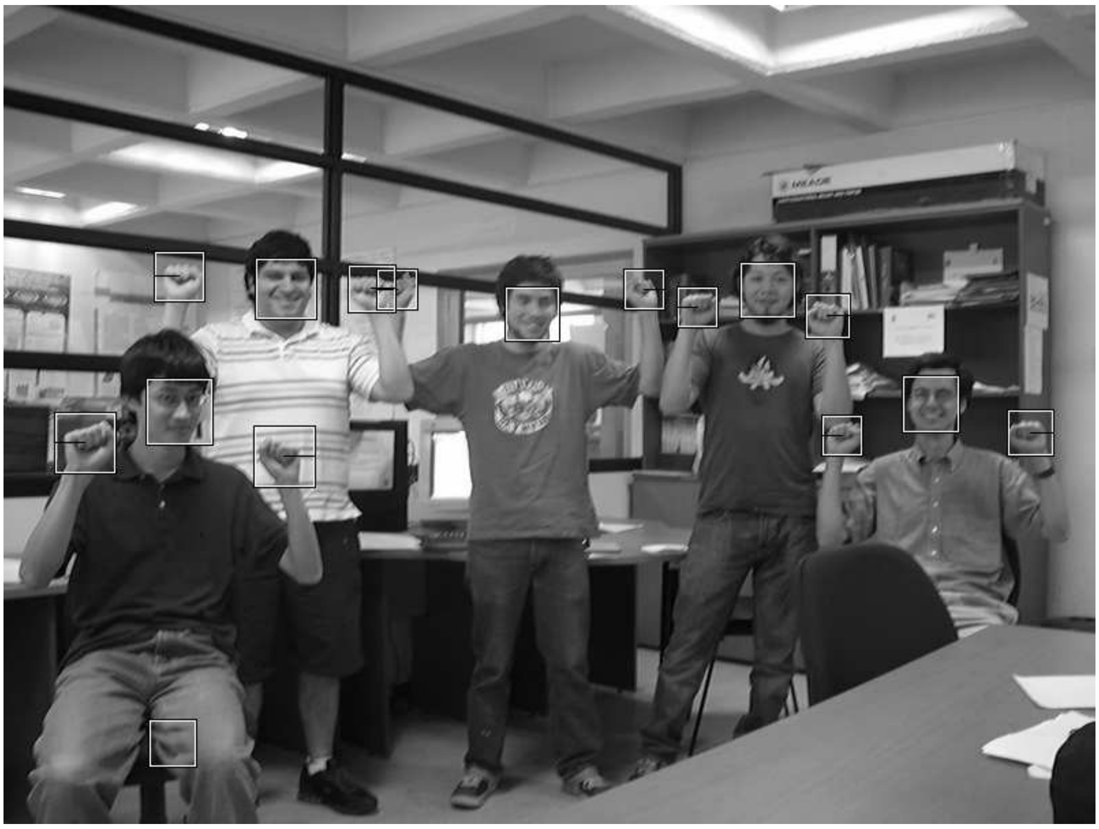


(a) HF75

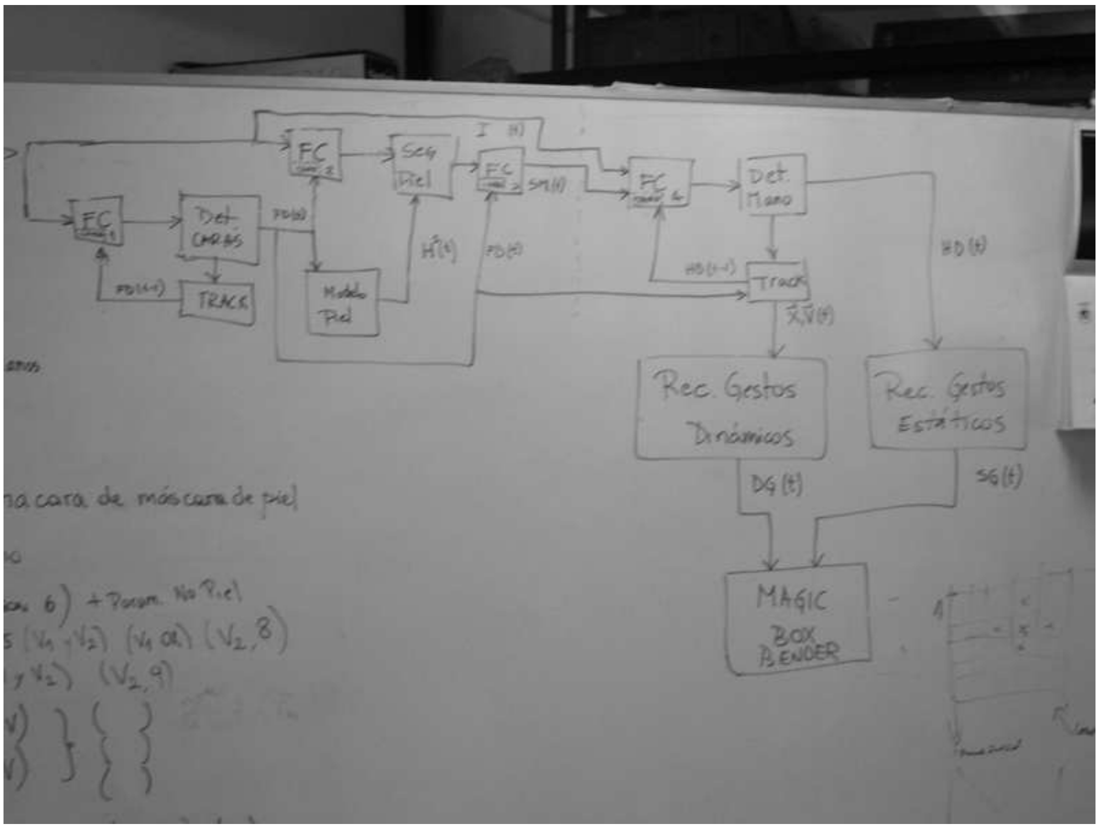


(b) HF75r

Figure 5.24: Hand and Face detection results. Problem: 3-classes (frontal left fist, frontal right fist and frontal faces).



(a) HF51



(b) HF34

Figure 5.25: Hand and Face detection results. Problem: 3-classes (frontal left fist, frontal right fist and frontal faces).

Chapter 6

Conclusions

The problem of object detection is central in the computer vision field. Even though there have been several advances in the last years, it is still an open problem. The most common example is the problem of face detection. The detection of objects is important for many applications, including human-computer interfaces, security, robotics, etc.

An approach that has proven to be effective for developing object detection systems is the use of cascade classifiers built using boosting algorithms, approach introduced by Viola and Jones that allows to obtain fast and robust detection results. The main idea of this approach is based on the fact that most windows (parts of the image) being analyzed do not contain the object. Therefore to obtain a fast detection, less processing time should be expended in parts of the image that do not resemble the object. This can be achieved, using a classifier with a cascade structure, where the first layers of the cascade quickly discard windows that are very different from the object. Viola and Jones proposed to use a boosting algorithm to train each layer of cascade, and also uses it for feature selection.

Taking the work of Viola and Jones as starting point, we have proposed methods for building one-class and multi-class object detectors. The contributions of this thesis work are basically three fold. First, an efficient learning framework of nested boosted cascades for one-class detection problems, with special emphasis on the training procedures, is proposed. Secondly, the proposed methods and the nested cascade are extended to the multi-class case. Finally the *TCAS* classifier is proposed, classifier that allows to better deal with complex multiclass object detection problems.

Most existing learning frameworks have a high computational complexity and require to predefine the structure of the classifier. The proposed learning framework for the one-class object detection problem is designed to build robust and fast object detectors in an efficient way without the need to predefine the structure of the classifier. The most interesting aspect of this framework is the integration of powerful learning capabilities together with effective training procedures, which allows building detection and classification systems that have high accuracy, robustness, and processing speed. In addition, the training time is considerably reduced compared with previous work. Using the proposed framework, a diversity of object of detection systems are built, and state of the art results are obtained. The built systems include a face detector, an eyes detector, a gender classifier, a profile car detector, hand detectors, and Aibo and humanoid robot detectors.

When building multiclass object detection systems, a problem arises when several object types are to be detected, case where both, the processing time and the false positive rates, may increase. Seeking to find a solution to this problem we extend the proposed nested boosted cascade classifiers and training algorithms to the multiclass case. The main goal is to reduce the computational burden of running several classifiers in parallel. For this, the use of different kinds of multiclass weak classifiers is analyzed, including two existing methods (independent components and joint components) and a new one – coupled components – that is here introduced. In addition we propose to perform a coarse-to-fine (CTF) search in the object target space, procedure that allows to considerable reduce the processing time and the number of false positive of the multiclass cascade. Using this methods CTF multiclass nested cascades are build and good results are obtained, showing that the methods scale relatively well with the number of classes in the training and detection processes.

To better deal with complex the multiclass detection problem, a new classifier structure, so called *TCAS*, is proposed. The *TCAS* classifier corresponds to a nested coarse-to-fine trees of multiclass boosted cascades. Results showing that the proposed system scales well with the number of classes, both during training and running time, are presented. We also show that is better to use coupled components in the first layers of the *TCAS* and independent components in the following ones. The proposed algorithms are evaluated by building a multiclass detector of hand fists and faces, and a multiview RIP face detector where state of the art results on the CMU rotated database are obtained . The obtained *TCAS*s present considerably shorter processing times and slightly better performance when compared with the use of specific cascades running in parallel. The presented results show the robustness of the proposed methods for building multiclass and multiview object detectors systems.

There is still room for improvement in the proposed methods. Some possible directions include: *i*) to explore the use of other kinds of features, *ii*) to define a criterion to select the feature type to be used at each level of the cascade/tcas, *iii*) to use LUT classifiers with partitions of variable size, *iv*) to combine the proposed *TCAS* classifier with the ideas of [Wu and Nevatia, 2007] to build jointly multiclass and multiview/pose systems *v*) to define a criterion to group the classes instead of using predefined groups when training a *TCAS* *vi*) to define a criterion to select the weak classifier training method to be used at each level of the cascade/*TCAS*, *vii*) to define a criterion to split a note when training the *TCAS*, *viii*) the use and design of codes to be used together with vectorboost, and *ix*) the use of target spaces in vectorboost not defined by inner products. Other open problems and possible research lines are: *i*) the incorporation of semi-supervised learning techniques in order to reduce the amount of work needed when building the training sets, *ii*) the use of context information, *iii*) the use of information about the relation between objects (e.g. relative position and scale), and *iv*) the use of transfer learning methods when learning a new detector or to re-train a detector adding new classes or views.

Bibliography

[Agarwal et al., 2004] Agarwal, S., Awan, A., and Roth, D. (2004). Learning to detect objects in images via a sparse, part-based representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1475–1490.

[Amit et al., 2004] Amit, Y., Geman, D., and Fan, X. (2004). A coarse-to-fine strategy for multiclass shape detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(12):1606–1621.

[Arenas et al., 2007] Arenas, M., del Solar, J. R., and Verschae, R. (2007). Detection of aibo and humanoid robots using cascades of boosted classifiers. In Visser, U., Ribeiro, F., Ohashi, T., and Dellaert, F., editors, *RoboCup*, volume 5001 of *Lecture Notes in Computer Science*, pages 449–456. Springer.

[Arenas et al., 2009] Arenas, M., Ruiz-del-Solar, J., and Verschae, R. (2009). Visual detection of legged robots and its application to robot soccer playing and refereeing.

[Baluja et al., 2004] Baluja, S., Sahami, M., and Rowley, H. A. (2004). Efficient face orientation discrimination. In *Int. Conf. on Image Processing*, pages 589–592.

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

[Bourdev and Brandt, 2005] Bourdev, L. and Brandt, J. (2005). Robust object detection via soft cascade. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) - Vol 2*, pages 236–243.

[Brubaker et al., 2006] Brubaker, S. C., Mullin, M. D., and Rehg, J. M. (2006). Towards optimal training of cascaded detectors. In *9th European Conf. on Computer Vision (ECCV 2006), LNCS 3951*, volume 2, pages 325–337.

[Brubaker et al., 2008] Brubaker, S. C., Wu, J., Sun, J., Mullin, M. D., and Rehg, J. M. (2008). On the design of cascades of boosted ensembles for face detection. *International Journal of Computer Vision*, 77(1-3):65–86.

[Buchala et al., 2004] Buchala, S., Davey, N., Frank, R. J., Gale, T. M., and Kanargard, M. J. L. W. (2004). Gender classification of face images: The role of global and feature-based information. In *Lecture Notes in Computer Science 3316*, pages 763–768. Publisher Springer Berlin / Heidelberg.

[Chen et al., 2006] Chen, T., Yin, W., Zhou, X. S., Comaniciu, D., and Huang, T. S. (2006). Total variation models for variable lighting face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1519–1524.

[Delakis and Garcia, 2004] Delakis, M. and Garcia, C. (2004). Convolutional face finder: A neural architecture for fast and robust face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1408–1423.

[Fasel et al., 2005] Fasel, I., Fortenberry, B., and Movellan, J. (2005). A generative framework for real time object detection and classification. *Computer Vision and Image Understanding*, 98:182–210.

[Fleuret and Geman, 2001] Fleuret, F. and Geman, D. (2001). Coarse-to-fine face detection. *International Journal of Computer Vision*, 41(1/2):85–107.

[Franke et al., 2007] Franke, H., del Solar, J. R., and Verschae, R. (2007). Real-time hand gesture detection and recognition using boosted classifiers and active learning. In Mery, D. and Rueda, L., editors, *PSIVT*, volume 4872 of *Lecture Notes in Computer Science*, pages 533–547. Springer.

[Freeman and Adelson, 1991] Freeman, W. T. and Adelson, E. H. (1991). The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):891–906.

[Freund and Mason, 1999] Freund, Y. and Mason, L. (1999). The alternating decision tree learning algorithm. In *Proc. 16th International Conf. on Machine Learning*, pages 124–133. Morgan Kaufmann, San Francisco, CA.

[Freund and Schapire, 1999] Freund, Y. and Schapire, R. (1999). A short introduction to boosting. *J. Japan. Soc. for Artificial Intelligence*, 14(5):771–780.

[Friedman et al., 1998] Friedman, J., Hastie, T., and Tibshirani, R. (1998). Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:2000.

[Fröba and Ernst, 2004] Fröba, B. and Ernst, A. (2004). Face detection with the modified census transform. In *Proc. 6th Int. Conf. on Face and Gesture Recognition*, pages 1–96.

[Fröba and Küblbeck, 2002] Fröba, B. and Küblbeck, C. (2002). Robust face detection at video frame rate based on edge orientation features. In *FGR '02: Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition*, Washington, DC, USA. IEEE Computer Society.

[Gangaputra and Geman, 2006] Gangaputra, S. and Geman, D. (2006). A design principle for coarse-to-fine classification. In *Proc. of the IEEE Conference of Computer Vision and Pattern Recognition*, volume 2, pages 1877–1884.

[Hermosilla et al., 2009] Hermosilla, G., Ruiz-del-Solar, J., Verschae, R., and Correa, M. (2009). Face recognition using thermal infrared images for human-robot interaction applications: A comparative study. In *LARS 2009, Oct. 29 - 30, Valparaso, Chile (CD Proceedings)*.

[Hjelms and Low, 2001] Hjelms, E. and Low, B. K. (2001). Face detection: A survey. *Computer Vision and Image Understanding*, 83(3):236–274.

[Huang et al., 2007] Huang, C., Ai, H., Li, Y., and Lao, S. (2007). High-performance rotation invariant multiview face detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(4):671–686.

[Huang et al., 2005] Huang, X., Li, S. Z., and Wang, Y. (2005). Jensen-Shannon boosting learning for object recognition. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 144–149. IEEE Computer Society.

[Jesorsky et al., 2001] Jesorsky, O., Kirchberg, K. J., and Frischholz, R. W. (2001). Robust face detection using the hausdorff distance. In *3rd Int. Conf. on Audio- and Video-based Biometric Person Authentication, (Lecture Notes in Computer Science, LNCS-2091)*, pages 90–95.

[Jones and Viola, 2003a] Jones, M. and Viola, P. (2003a). Face recognition using boosted local features. In *Proceedings of International Conference on Computer Vision*.

[Jones and Viola, 2003b] Jones, M. and Viola, P. (2003b). Fast multi-view face detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Keren et al., 2001] Keren, D., Osadchy, M., and Gotsman, C. (2001). Antifaces: A novel, fast method for image detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(7).

[Kienzle et al., 2004] Kienzle, W., Bakir, G. H., Franz, M. O., and Schölkopf, B. (2004). Efficient approximations for support vector machines in object detection. In *Lecture Notes in Computer Science*, volume 3175, pages 54–61.

[Kirchberg et al., 2002] Kirchberg, K. J., Jesorsky, O., and Frischholz, R. W. (2002). Genetic model optimization for hausdorff distance-based face localization. In *Int. ECCV 2002 Workshop on Biometric Authentication (Lecture Notes In Computer Science; Vol. 2359)*, pages 103–111.

[Kölsch and Turk, 2004] Kölsch, M. and Turk, M. (2004). Robust hand detection. In *Proceedings of the sixth International Conference on Automatic Face and Gesture Recognition*, pages 614–619.

[Lampert et al., 2009] Lampert, C. H., Blaschko, M. B., and Hofmann, T. (2009). Efficient subwindow search: A branch and bound framework for object localization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31:2129–2142.

[Larlus et al., 2009] Larlus, D., Verbeek, J., and Jurie, F. (2009). Category level object segmentation by combining bag-of-words models with dirichlet processes and random fields. *International Journal of Computer Vision*.

[Lee-Ferng et al., 2010] Lee-Ferng, J., Ruiz-del-Solar, J., Correa, M., and Verschae, R. (2010). Hand gesture recognition for human robot interaction in uncontrolled environments. In *Workshop on Multimodal Human - Robot Interfaces, 2010 IEEE International Conference on Robotics and Automation May 3, 2010, Anchorage, Alaska, (Accepted)*.

[Leibe et al., 2004] Leibe, B., Leonardis, A., and Schiele, B. (2004). Combined object categorization and segmentation with an implicit shape model. In *In ECCV workshop on statistical learning in computer vision*, pages 17–32.

[Li and Zhang, 2004] Li, S. Z. and Zhang, Z. (2004). Floatboost learning and statistical face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1112–1123.

[Lienhart et al., 2003] Lienhart, R., Kuranov, A., and Pisarevsky, V. (2003). Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *Proc. DAGM-Symposium*.

[Lin et al., 2004] Lin, Y.-Y., Liu, T.-L., and Fuh, C.-S. (2004). Fast object detection with occlusion. In *Proc. 8th European Conference on Computer Vision (Lecture Notes in Computer Science 3021)*, pages 402–413.

[Liu and Shum, 2003] Liu, C. and Shum, H.-Y. (2003). Kullback-leibler boosting. In *Proc. of the IEEE Conference of Computer Vision and Pattern Recognition*, pages 587–594.

[Liu and Nocedal, 1989] Liu, D. and Nocedal, J. (1989). On the limited memory method for large scale optimization. *Mathematical Programming B*, 45(3):503–528.

[Ma et al., 2004] Ma, Y., Ding, X., Wang, Z., and Wang, N. (2004). Robust precise eye location under probabilistic framework. In *Proc. 6th International Conference on Face and Gesture Recognition*, pages 339–344.

[Meynet et al., 2007] Meynet, J., Popovici, V., and Thiran, J.-P. (2007). Face detection with boosted gaussian features. *Pattern Recognition*, 40(8):2283–2291.

[Moghaddam and Yang, 2002] Moghaddam, B. and Yang, M.-H. (2002). Learning gender with support faces. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):707–711.

[Moosmann et al., 2008] Moosmann, F., Nowak, E., and Jurie, F. (2008). Randomized clustering forests for image classification. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(9):1632–1646.

[Mutch and Lowe, 2006] Mutch, J. and Lowe, D. G. (2006). Multiclass object recognition with sparse, localized features. In *CVPR (1)*, pages 11–18. IEEE Computer Society.

[Mutch and Lowe, 2008] Mutch, J. and Lowe, D. G. (2008). Object class recognition and localization using sparse features with limited receptive fields. *International Journal of Computer Vision (IJCV)*, 80(1):45–57.

[Osuna et al., 1997] Osuna, E., Freund, R., and Girosi, F. (1997). Training support vector machines: an application to face detection. In *Proc. of the IEEE Conference of Computer Vision and Pattern Recognition*, pages 130–136.

[Papageorgiou and Poggio, 2000] Papageorgiou, C. and Poggio, T. (2000). A trainable system for object detection. *International Journal of Computer Vision*, 38(1):15–33.

[Phillips et al., 1998] Phillips, P. J., Wechsler, H., Huang, J., and Rauss, P. J. (1998). The feret database and evaluation procedure for face-recognition algorithms. *Image Vision Comput.*, 16(5):295–306.

[Ponce et al., 2006] Ponce, J., Berg, T. L., Everingham, M., Forsyth, D. A., Hebert, M., Lazebnik, S., Marszalek, M., Schmid, C., Russell, B. C., Torralba, A., Williams, C. K. I., Zhang, J., and Zisserman, A. (2006). Dataset issues in object recognition. In *Toward Category-Level Object Recognition, volume 4170 of LNCS*, pages 29–48. Springer.

[Romdhani et al., 2001] Romdhani, S., Torr, P., Schölkopf, B., and Blake, A. (2001). Computationally efficient face detection. In *Proceeding of the 8th International Conference on Computer Vision*.

[Rowley et al., 1998] Rowley, H. A., Baluja, S., and Kanade, T. (1998). Neural network-based detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–28.

[Ruiz-del-Solar et al., 2009a] Ruiz-del-Solar, J., Mascaró, M., Correa, M., Bernuy, F., Riquelme, R., and Verschae, R. (2009a). Analyzing the human-robot interaction abilities of a general-purpose social robot in different naturalistic environments. In *Lecture Notes in Computer Science (RoboCup Symposium 2009) (in press)*.

[Ruiz-del-Solar and Navarrete, 2005] Ruiz-del-Solar, J. and Navarrete, P. (2005). Eigenspace-based face recognition: A comparative study of different approaches. *IEEE Transactions on Systems, Man, and Cybernetics C (Special Issue on Biometric Systems)*, 35(3):315–325.

[Ruiz-del-Solar et al., 2009b] Ruiz-del-Solar, J., Verschae, R., and Correa, M. (2009b). Recognition of faces in unconstrained environments: A comparative study. *EURASIP Journal on Advances in Signal Processing*, 2009.

[Ruiz-del-Solar et al., 2009c] Ruiz-del-Solar, J., Verschae, R., Correa, M., Lee-Ferng, J., and Castillo, N. (2009c). Real-time hand gesture recognition for human robot interaction. lecture notes in computer science (robocup symposium 2009) (in press).

[Ruiz-del-Solar et al., 2009d] Ruiz-del-Solar, J., Verschae, R., Lee-Ferng, J., and Correa, M. (2009d). Dynamic gesture recognition for human robot interaction. In *LARS 2009, Oct. 29 - 30, Valparaso, Chile (CD Proceedings)*.

[Ruiz-del-Solar et al., 2007] Ruiz-del-Solar, J., Verschae, R., Vallejos, P., and Correa, M. (2007). Face analysis for human computer interaction applications. In Ranchordas, A., Araújo, H., and Vitrià, J., editors, *VISAPP (Special Sessions)*, pages 23–30. INSTICC - Institute for Systems and Technologies of Information, Control and Communication.

[Sahbi and Geman, 2006] Sahbi, H. and Geman, D. (2006). A hierarchy of support vector machines for pattern detection. *Journal of Machine Learning Research*, 7:2087–2123.

[Schapire and Singer, 1999] Schapire, R. and Singer, Y. (1999). Improved boosting using confidence-rated predictions. *Machine Learning*, 37(3):297–336.

[Schneiderman, 2004] Schneiderman, H. (2004). Feature-centric evaluation for efficient cascade object detection. In *Proc. of the IEEE Conference of Computer Vision and Pattern Recognition*, pages 29–36.

[Schneiderman and Kanade, 2000] Schneiderman, H. and Kanade, T. (2000). A statistical model for 3D object detection applied to faces and cars. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 746–751.

[Shakhnarovich et al., 2002] Shakhnarovich, G., Viola, P. A., and Moghaddam, B. (2002). A unified learning framework for real time face detection and classification. In *FGR '02: Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition*, pages 14–21, Washington, DC, USA. IEEE Computer Society.

[Sun et al., 2004] Sun, J., Rehg, J. M., and Bobick, A. F. (2004). Automatic cascade training with perturbation bias. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004)*, volume 2, pages 276–283.

[Sung and Poggio, 1998] Sung, K.-K. and Poggio, T. (1998). Example-based learning for viewed-based human face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):39–51.

[Todorovic and Ahuja, 2006] Todorovic, S. and Ahuja, N. (2006). Extracting subimages of an unknown category from a set of images. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 927–934.

[Torralba et al., 2004a] Torralba, A., Murphy, K. P., and Freeman, W. T. (2004a). Shared features for multiclass object detection. In *Proc. of the Workshop: Towards Category-Level Object Recognition. Lecture Notes in Computer Science (LNCS 4170)*.

[Torralba et al., 2004b] Torralba, A., Murphy, K. P., and Freeman, W. T. (2004b). Sharing features: efficient boosting procedures for multiclass object detection. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 762–769.

[Torralba et al., 2007] Torralba, A., Murphy, K. P., and Freeman, W. T. (2007). Sharing visual features for multiclass and multiview object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(5):854–869.

[Verschae, 2006] Verschae, R. (2006). Improvement of a face detection system using A-contrario validation methods. Technical report, CMLA, ENS Cachan.

[Verschae and Ruiz-del-Solar, 2003] Verschae, R. and Ruiz-del-Solar, J. (2003). A hybrid face detector based on an asymmetrical adaboost cascade detector and a wavelet-bayesian-detector. In *Lecture Notes in Computer Science 2686*, pages 742–749.

[Verschae and Ruiz-del-Solar, 2006] Verschae, R. and Ruiz-del-Solar, J. (2006). *Applications of Soft Computing: Recent Trends*, volume 36, chapter State of the Art Face Detection: Cascade Boosting Approaches. Springer.

[Verschae and Ruiz-del-Solar, 2008] Verschae, R. and Ruiz-del-Solar, J. (2008). Multiclass adaboost and coupled classifiers for object detection. In Ruiz-Shulcloper, J. and Kropatsch, W. G., editors, *CIARP*, volume 5197 of *Lecture Notes in Computer Science*, pages 560–567. Springer.

[Verschae et al., 2006] Verschae, R., Ruiz-del-Solar, J., and Correa, M. (2006). Gender classification of faces using adaboost. In *Lecture Notes in Computer Science 4225 (CIARP 2006)*, pages 68–77.

[Verschae et al., 2008a] Verschae, R., Ruiz-Del-Solar, J., and Correa, M. (2008a). Face Recognition in Unconstrained Environments: A Comparative Study. In *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition*, Marseille France. Erik Learned-Miller and Andras Ferencz and Frédéric Jurie.

[Verschae et al., 2008b] Verschae, R., Ruiz-del-Solar, J., and Correa, M. (2008b). A unified learning framework for object detection and classification using nested cascades of boosted classifiers. *Machine Vision Applications*, 19(2):85–103.

[Verschae et al., 2005] Verschae, R., Ruiz-del-Solar, J., Köppen, M., and Vicente-Garcia, R. (2005). Improvement of a face detection system by evolutionary multi-objective optimization. In *Proc. 5th Int. Conf. on Hybrid Intelligent Systems*, pages 361–366.

[Vidal-Naquet and Ullman, 2003] Vidal-Naquet, M. and Ullman, S. (2003). Object recognition with informative features and linear classication. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*.

[Villamizar et al., 2006] Villamizar, M., Sanfeliu, A., and Andrade-Cetto, J. (2006). Orientation invariant features for multiclass object recognition. In *Lecture Notes in Computer Science 4225 (CIARP 2006)*, pages 655–664.

[Viola and Jones, 2001] Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 511–518.

[Viola and Jones, 2002] Viola, P. and Jones, M. (2002). Fast and robust classification using asymmetric adaboost and a detector cascade. In *Advances in Neural Information Processing System 14*. MIT Press.

[Viola et al., 2003] Viola, P., Jones, M., and Snow, D. (2003). Detecting pedestrians using patterns of motion and appearance. In *Proc. 9th IEEE Int. Conf. on Computer Vision*, volume 2, pages 734–741.

[Viola and Jones, 2004] Viola, P. and Jones, M. J. (2004). Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154.

[Wang et al., 2004] Wang, H., Li, S. Z., and Wang, Y. (2004). Face recognition under varying lighting conditions using self quotient image. In *Sixth IEEE International Conference on Automatic Face and Gesture Recognition*, pages 819–824, Los Alamitos, CA, USA. IEEE Computer Society.

[Wu et al., 2003a] Wu, B., Ai, H., and Huang, C. (2003a). Lut-based adaboost for gender classification. In *Lecture Notes in Computer Science 2688*, pages 104–110. Springer Berlin / Heidelberg.

[Wu et al., 2004] Wu, B., AI, H., Huang, C., and Lao, S. (2004). Fast rotation invariant multiview face detection based on real adaboost. In *Proc. of the 6th Int. Conf. on Face and Gesture Recognition*, pages 79–84.

[Wu and Nevatia, 2005] Wu, B. and Nevatia, R. (2005). Detection of multiple, partially occluded humans in a single image by bayesian combination of edgelet part detectors. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, pages 90–97, Washington, DC, USA. IEEE Computer Society.

[Wu and Nevatia, 2007] Wu, B. and Nevatia, R. (2007). Cluster boosted tree classifier for multiview, multi-pose object detection. In *Proceedings of the Eleventh IEEE International Conference on Computer Vision (ICCV'07)*. IEEE Computer Society.

[Wu et al., 2003b] Wu, J., Rehg, J. M., and Mullin, M. D. (2003b). Learning a rare event detection cascade by direct feature selection. In *Proc. Advances in Neural Information Processing Systems 16*. MIT Press.

[Xiao et al., 2003] Xiao, R., Zhu, L., and Zhang, H.-J. (2003). Boosting chain learning for object detection. In *Proc. 9th IEEE Int. Conf. on Computer Vision*, volume 1, pages 709–715.

[Yang et al., 2000a] Yang, M.-H., Ahuja, N., and Kriegman, D. (2000a). Mixtures of linear subspaces for face detection. In *Proc. Fourth IEEE Int. Conf. on Automatic Face and Gesture Recognition*, pages 70–76.

[Yang et al., 2002] Yang, M.-H., Kriegman, D., and Ahuja, N. (2002). Detecting faces in images: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(1):34–58.

[Yang et al., 2000b] Yang, M.-H., Roth, D., and Ahuja, N. (2000b). A SNoW-based face detector. *Advances in Neural Information Processing Systems 12*, pages 855–861.