Multivariate Boosting with Look-Up Tables for Face Processing

THÈSE N^{O} 5374 (2012)

PRÉSENTÉE LE 22 JUIN 2012

À LA FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR

LABORATOIRE DE L'IDIAP

PROGRAMME DOCTORAL EN GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Cosmin ATANASOAEI

acceptée sur proposition du jury:

Prof. J.-Ph. Thiran, président du jury
Prof. H. Bourlard, Dr S. Marcel, directeurs de thèse
Prof. T. Cootes, rapporteur
Prof. J. Kittler, rapporteur
Dr V. Lepetit, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2012

Abstract

This thesis proposes a novel unified boosting framework. We apply this framework to the several face processing tasks, face detection, facial feature localisation, and pose classification, and use the same boosting algorithm and the same pool of features (local binary features). This is in contrast with the standard approaches that make use of a variety of features and models, for example AdaBoost, cascades of boosted classifiers and Active Appearance Models.

The unified boosting framework covers multivariate classification and regression problems and it is achieved by interpreting boosting as optimization in the functional space of the weak learners. Thus a wide range of smooth loss functions can be optimized with the same algorithm. There are two general optimization strategies we propose that extend recent works on TaylorBoost and Variational AdaBoost. The first proposition is an empirical expectation formulation that minimizes the average loss and the second is a variational formulation that includes an additional penalty for large variations between predictions.

These two boosting formulations are used to train real-time models using local binary features. This is achieved using look-up-tables as weak learners and multi-block Local Binary Patterns as features. The resulting boosting algorithms are simple, efficient and easily scalable with the available resources. Furthermore, we introduce a novel coarse-to-fine feature selection method to handle high resolution models and a bootstrapping algorithm to sample representative training data from very large pools of data.

The proposed approach is evaluated for several face processing tasks. These tasks include frontal face detection (binary classification), facial feature localization (multivariate regression) and pose estimation (multivariate classification). Several studies are performed to assess different optimization algorithms, bootstrapping parametrizations and feature sharing methods (for the multivariate case). The results show good performance for all of these tasks.

In addition to this, two other contributions are presented. First, we propose a context-based model for removing the false alarms generated by a given generic face detector. Second, we propose a new face detector that predicts the Jaccard distance between the current location and the ground truth. This allows us to formulate the face detection problem as a regression task.

Keywords: Boosting, look-up tables, multi-block Local Binary Patterns, bootstrapping, coarse-to-fine feature selection, face detection, facial feature localization, pose estimation.

Résumé

Dans cette thèse, nous proposons une nouvelle approche pour le traitement du visage impliquant l'apprentisage par dopage ("boosting") de motifs binaires locaux (LBP). L'approche proposée aborde certaines sous-tâches spécifiques en traitement des visages avec un algorithme de "boosting" et un jeux de caractéristiques identiques. Ceci est en contraste avec les approches standard qui utilisent une variété de fonctionnalités et de modèles, comme par exemple AdaBoost, des cascades et des modèles d'apparence actifs (Active Shape Models).

Nous proposons un cadre unifié pour l'apprentissage par "boosting". Ce cadre unifié couvre la classification multivariée et les problèmes de régression, et ceci est réalisé en interprétant le "boosting" comme une optimisation dans l'espace fonctionel d'algorithmes d'apprentissage faible ("weak learner"). Ainsi, un large éventail de fonctions de coût lisses ("smooth loss functions") peuvent être optimisées avec le même algorithme. Il existe deux stratégies d'optimisation générale que nous proposons qui s'étendent des travaux récents sur TaylorBoost et AdaBoost Variationnel. La première proposition est une formulation empiriques de l'espérance qui minimise la moyenne du coût, et la seconde est une formulation variationnelle qui comprend une pénalité supplémentaire pour de grandes variations entre les prédictions.

Ces deux formulations sont utilisés pour entrainer des modèles temps-réel en utilisant les motifs binaires locaux (LBP). Ceci est réalisé en utilisant des tables de correspondance (look-up tables ou LUT) comme d'algorithmes d'apprentissage faibles et des motifs binaires locaux multi-blocs comme caractéristiques. Les algorithmes de "boosting" obtenus sont simples, efficaces et facilement évolutif ("scalable") avec les ressources disponibles. En outre, nous introduisons une nouvelle méthode de sélection des caractéristiques dites "coarse-to-fine" pour gérer les modèles à haute résolution et un algorithme d'amorçage ("bootstrapping") pour échantillonner des données d'entrainement représentative dans de très grandes quantités de données.

L'approche proposée est évaluée pour plusieurs tâches de traitement du visage. Ces tâches incluent la détection de visage de face (classification binaire), la localisation de caractéristiques faciales sur le visage (régression multivariée) et estimation de la pose (classification multivariée). Plusieurs études sont effectuées pour évaluer différents algorithmes d'optimisation, les paramétrages de l'algorithme d'amorçage ("bootstrapping"), et les méthodes de partage des caractéristiques (pour le cas multivarié). Les résultats montrent une bonne performance pour toutes ces tâches.

En plus de cela, deux autres contributions sont présentées. Tout d'abord, nous proposons un modèle basé sur le contexte pour éliminer les fausses alarmes générées par un détecteur de visage générique donné. Deuxièmement, nous proposons un nouveau détecteur de visage qui prédit la distance de Jaccard entre l'emplacement actuel et la vérité terrain ("ground truth"). Cela permet de formuler le problème de détection de visage comme une tâche de régression.

Mots-clés : Apprentisage par dopage ("boosting"), tables de correspondances ("look-up tables"), motifs binaires locaux multi-blocs, algorithmes d'amorçage ("bootstrapping"), détection de visage, localisation de caractéristiques faciales, estimation de la pose.

Acknowledgements

It is a good morning exercise for a research scientist
to discard a pet hypothesis every day before breakfast.
It keeps him young.

- Konrand Lorenz, *1903-1989*

This PhD was an unforgettable experience. There are several people who guide me through this experience. Firstly, Sébastien and Chris, my supervisor and co-supervisor, have greatly contributed with their constant help and fruitful discussions to polish, improve and crystallize my ideas into this thesis. I would also like to thank Hervé, my thesis director, and to all the jury members - Prof. Tim Cootes, Prof. Joseph Kittler, Dr. Vincent Lepetit and Prof. Jean-Philippe Thiran.

Then there is the brainstorming group of Niklas, Hugo, Leo and Charles for innumerable discussions and bashing of various crazy machine learning and computer vision ideas. A special thanks to Francesco for smoothing my baby steps into machine learning. Nonetheless I would like to thank to all my colleagues and friends from Idiap, especially for the fun mid-day baby-foot sessions and the coffee/tea discussions.

The administrative and technical staff was always prompt to help me whenever needed. I would like to thank for this to all colleagues from Idiap and EPFL, to Nadine, Sylvie, Corinne, Norbert, Frank, Bastien and to all the system guys and the developers.

Last but not the least, I should mention the constant support and encouragement of Corina and of my parents. I cannot thank them enough for all that they have done for me.

Contents

List of Figures

List of Tables

Glossary and acronyms

LUT	Look-Up Table
MB	Multi-Block
LBP	Local Binary Pattern
MCT	Modified Census Transform
MB-LBP	Multi-Block Local Binary Pattern
MB-MCT	Multi-Block Modified Census Transform
EMB-LBP	Extended set of Multi-Block Local Binary Pattern
CTFFS	Coarse-To-Fine Feature Selection
SHARED	Model trained using Shared feature selection
INDEP	Model trained using Independent feature selection
BOOT	Bootstrapped model
SHOT	Boosted model without bootstrapping
EPT	Model trained using the Expectation loss formulation
VAR	Model trained using the Variational loss formulation

Chapter 1

Introduction

Face processing is a mature research field, being under active research for two decades. Great progress has been made, such that successful applications exist on computers or mobile devices. For example, face detection is implemented in video cameras and automatic biometric authentication, surveillance, tracking and security systems have been already deployed successfully. This thesis proposes a novel approach to further improve such systems.

1.1 Objective of the thesis

A typical face processing system usually consists of an ad-hoc mix of features and machine learning algorithms. For example, most successful *face detectors* use Haar (Viola and Jones, 2001) or Local Binary Patterns (Zhang et al., 2007) as features and boosted cascades trained using a variety of methods. By contrast, *facial feature localization* is usually performed using Active Shape Models and Active Appearence Models (Cristinacce and Cootes, 2006, 2007, 2008). The list of successful techniques is definitely not exhausted.

Most of the enumerated methods run in real-time and are robust to some degree of pose and illumination variation. The computational efficiency becomes hard to achieve when several such methods are combined to address specific face processing problems. This is because different classes of features and machine learning algorithms usually require different pre-processing steps which are time consuming.

The objective of this thesis is to address the computational efficiency problem of face processing. We propose a unified boosting framework to address several face processing tasks. The boosting framework can be used: i) for multivariate classification and regression tasks, and ii) can efficiently use large amounts of training data and high resolution models. In particular, we present experimental results for these face processing tasks: frontal face detection, facial feature localization and pose estimation.

1.2 Motivations

We propose to extend recent work on boosting in two ways. First, we propose a unified boosting framework that can be applied to multivariate classification and regression tasks. Second, we propose an efficient method to sample from large pools of samples and an efficient method to select relevant features for high resolution models.

The first proposal consists of a novel multivariate boosting method for classification and regression problems. Our approach extends recent boosting algorithms such as AnyBoost (Mason et al., 1999b), TaylorBoost (Masnadi-Shirazi, 2010), Empirical Bernstein Boosting (Shivaswamy and Jebara, 2010) and Variational AdaBoost (Shivaswamy and Jebara, 2011). Similarly, we interpret boosting as a gradient descent algorithm in the functional space of weak learners. Thus a large set of smooth loss functions can be efficiently optimized.

The second proposal consists of a generic method to boost high resolution models using very large pools of samples. First, we introduce a novel coarse-to-fine feature selection method to efficiently select generic multi-block patterns (Zhang et al., 2007; Trefny and Matas, 2010). This is particularly useful for high resolution models, when the number of available features is of the order of millions. Second, we present a method to sample representative training data from very large pools of samples, for example of the order of billions. This is achieved in two ways: by uniformly sampling to obtain balanced data and by bootstrapping the training data with the missed predicted samples.

In this work we concentrate on boosting features that result in look-up tables (LUTs). As such, we detail several simple methods of boosting LUTs as weak learners. The optimization algorithm is proven to be scalable with the available computational resources at training time, with respect

to the number of samples (via bootstrapping) and features (via feature selection). At test time, we achieve real-time performance on several face processing tasks, such as: frontal face detection, facial feature localization and pose estimation. This is a step towards a simpler and more homogeneous face processing system.

1.3 Contributions

The major contributions of this thesis are as follows.

1. We proposed **a unified boosting framework to solve multivariate regression and classification problems**. The fundamental idea of this approach is to interpret boosting as a gradient descent method in the functional space of the weak learners. This way a wide range of smooth loss functions can be iteratively optimized. The extension to multivariate classification and regression problems becomes natural.

   We present a generic boosting framework that requires very few assertions on the loss function to optimize. The assumptions are that the loss function must be smooth and its gradients computable at each boosting round, with respect to the weak learner's parameters. Using this we formulate two boosting algorithms - the expectation and the variational. The **expectation** algorithm optimizes the empirical average loss over the samples, while the **variational** algorithm regularizes the average loss with the empirical variance of the same loss. The latter algorithm penalizes models that have high error variance, for example that perform well for some subset of samples and significantly worse for others.

2. We present an efficient implementation of our unified boosting framework for boosting LUTs. We focus on **boosting LUTs** because the resulting algorithm is efficient and scalable with the available computational resourses. Also, LUTs are the most efficient weak learners at test time, although they are non-linear and have a significant number of parameters. **Feature selection** is performed while boosting, because each boosted LUT is associated with a single feature. We also study two **feature sharing** methods for the multivariate case. Detailed complexity analysis is provided to assert the efficiency of the LUT boosting algorithm. Furthermore, boosting shared features for different outputs is shown to be of equivalent com-

plexity as boosting independent features.

3. Another contribution related to the proposed boosting algorithm consists of a **generic method for efficiently boosting high resolution models using large pools of samples**. This is particularly useful for face processing tasks when either the number of features is unmanageable because we have high resolution models, or because the number of samples to process is very large as we have to simulate minor mis-alignments for each training sample. For the first issue we propose a **coarse-to-fine** approach and for the second we propose to use **bootstrapping**.

   The key idea of the proposed coarse-to-fine multi-block feature selection is to process a small subset of all available features. The algorithm is initialized with the boosted features using a coarse resolution model. Then this subset is iteratively projected and boosted to higher resolutions. Each iteration refines the scale and the size of the current set of features.

   The fundamental idea of bootstrapping is to increase the number of training samples at each iteration with the missed predicted samples. At each step, the model is evaluated efficiently on a large pool of samples and an error is computed for each sample. Next, a small fixed number of samples are randomly selected proportionally with this error. The selected samples are also constrained to be balanced: for example, to contain the same number of positive and negative samples for binary classification problems.

4. The proposed boosted models are evaluated on several **face processing tasks**. In particular we present real-time frontal face detection, facial feature localization and pose estimation systems. These systems are extensively evaluated using different boosting configurations.

   To speed-up face detection, we propose to split the boosted model at run-time into **levels**. The number of levels influences the evaluation speed; the more, the faster the face detector. This way the background locations are discarded quickly, while the computation is concentrated on the most promising locations. This idea is similar to the successful boosted cascades. The difference is that our models are significantly easier to train than cascades, while experimentally performing as fast at test time. We also show experimentally that the proposed models are robust to the number of splits.

   The facial feature localization system makes use of boosted regression models that predict

the location of several facial features of interest (such as the eyes, nose tip or mouth corners). We use the proposed coarse-to-fine feature selection algorithm to efficiently train these high resolution models. The localization performance is increased by integrating their responses in a local neighbourhood. We propose and evaluate several integration methods.

The pose estimation system classifies a face detection into 13 different out-of-plane poses. We train and evaluate a multivariate classification model to distinguish between these 13 poses.

Apart from these primary contributions, there are some secondary contributions of this thesis which are related to the main work. These are as follows.

1. Firstly, we proposed a **method to remove false alarms generated by a generic face detector**. We define the context of a detection as the collection of nearby responses, defined in a tri-dimensional grid of location and scale. Several features are extracted from this collection of responses which are then used to train a linear classifier to distinguish between the context of a false alarm and of a true detection. For example, initial experiments have shown that false alarms have contexts with higher variance of responses and with lower number of accepted responses (above a given threshold) than for the true detections.

   Several variations of the proposed system were compared with the given face detector on multiple challenging image datasets. The experimental results have shown that it greatly reduces the number of false alarms, while keeping the detection rate at the same level.

2. Secondly, we proposed a novel face detection method that uses the **Jaccard similarity index to guide the detection**. A boosted regression model was trained to predict the Jaccard similarity index (normalized intersection area) between the current location and the true face location. Ideally, the model will predict 1 for locations close to the face and 0 for background locations. This information is used to guide the detection in two steps. The first is to coarsely initialize a set of potential locations and the second to refine the most promising locations - the ones with high scores.

   The advantage of such a method is to greatly speed-up the detection. The initial experiments have shown that it is significantly faster than sliding-window approaches. However, the predictions of the Jaccard similarity index are not accurate enough to reach state-of-the-art performance in face detection.

Related publications:

1. Cosmin Atanasoaei, Christopher McCool and Sébastien Marcel, "A principled approach to remove false alarms by modelling the context of a face detector", in proceedings of the British Machine Vision Conference (BMVC), 2010

2. Sébastien Marcel, Christopher McCool, Cosmin Atanasoaei, Flavio Tarsetti, J. Pesán, P. Matejka, J. Cernocky, M. Helistekangas, and M. Turtinen, "MOBIO: Mobile biometric face and speaker authentication", in proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010

3. J. Parris, M. Wilber, B. Helfin, H. Rara, A. El-barkouky, A. Farag, J. Movellan, M. Santana, J. Lorenzo, M.N. Teli, S. Marcel, C. Atanasoaei, and T. Boult, "Face and eye detection on hard datasets", in the IEEE IAPR International Joint Conference on Biometrics (IJCB), 2011

Related technical reports:

1. Cosmin Atanasoaei, Christopher McCool and Sébastien Marcel, "On Improving Face Detection Performance by Modelling Contextual Information", Idiap Research Report Idiap-RR-43-2010, Idiap Research Institute, December 2010

2. Cosmin Atanasoaei, Christopher McCool and Sébastien Marcel, "Face detection using boosted Jaccard distance-based regression", Idiap Research Report Idiap-RR-02-2012, Idiap Research Institute, January 2012

1.4 Organization

The structure of this thesis is as follows.

- Chapter 2 provides a brief overview of the related work to boosting and local binary features. First, we provide an overview of boosting and how it can be viewed as functional gradient descent. Then we describe local binary features including the extended set of local binary patterns.

- Chapter 3 details the proposed boosting approach of look-up-tables. First, we introduce the multivariate boosting framework and a bootstrapping algorithm. Second, we detail the optimization algorithms for boosting look-up-tables as weak learners.

- Chapter 4 describes the proposed coarse-to-fine feature selection and the bootstrapping algorithms to boost high resolution models using large pools of data.

- Chapter 5 describes the application of the proposed approach to the task of face detection, a binary classification problem. Different features, boosting algorithms and evaluation speeding-up methods are compared with baseline approaches.

- Chapter 6 describes the application of the proposed approach to the task of facial feature localization, a multivariate regression problem. The experiments evaluate and compare different features, feature sharing methods and boosting algorithms.

- Chapter 7 describes the application of the proposed approach to the task of face pose classification, a multivariate classification problem. The experiments evaluate and compare different features, feature sharing methods and boosting algorithms.

- Chapter 8 concludes the thesis with a brief summary of the important contributions made and outlines the potential directions for future work.

- In the Appendices, we describe some of the secondary contributions of this thesis. These include the work on face detection using the predicted Jaccard distance between the current sub-window and the ground truth (Appendix A) and the work on the context-based model for removing false alarms (Appendix B).

Chapter 2

Related work

Most computer vision applications require two key components. The first component is the **feature extraction** that maps the visual input signal (the pixel values) to a feature space. The features are usually designed to suppress irrelevant characteristics of the input signal, while enhancing the important characteristics for the task at hand. For example, some features are design to be invariant (or robust) to translation and rotation, illumination variation or to pose variation. The second component is the **modeling** which consists of training and evaluating a model using machine learning and statistical algorithms. There can also be some post-processing steps which are specific to the application.

This chapter provides an introduction to **boosting** - as a particular machine learning algorithm, and to **local binary patterns** - as particular features. These two concepts are the building blocks of the proposed approach.

2.1 Boosting

Boosting (Schapire, 2002) is a greedy algorithm for building a **strong learner** as a linear combination of **weak learners** or **hypotheses**. This process is done in iterations called **boosting rounds**: at each iteration a single new weak learner is chosen and added to the combination. The weak learners can be interpreted as rules of thumb, usually easy and efficient to construct. They are not accurate enough to solve the task at hand, but it is assumed that a more accurate model can

be built by combining such simple rules. Boosting thus provides an efficient method for building strong learners from many weak learners.

There are several methods to select a new weak learner at each round. This section details some of the most successful boosting methods. Their common idea is that each weak learner is trained to correct the mistakes made by, so as to complement, the previous ones and to focus on the most challenging samples.

2.1.1 Introduction

Let $\mathbf{\Omega}$ be the input signal space and $\{(\mathbf{x}_n, y_n)_{n=1:N}\} \in (\mathbf{\Omega} \times \mathbb{R})^N$ a set of N training samples. The goal is to build a functional $f : \mathbf{\Omega} \to \mathbb{R}$ to map the samples $\mathbf{x}_n$ to their targets y_n. We refer to this functional as the strong learner or the **model**.

Let $L(f)$ be the criteria (loss function) to choose the strong learner f. Usually, the criteria is a cumulative loss function of the form: $L(f) = \sum_n l(y_n, f(\mathbf{x}_n))$. The base loss $l(y, f)$ is a function that measures the goodness of the prediction f in matching the target y: $l(y, f) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. Clearly the loss must be chosen appropriately to the specific problem to solve. For example, the binary classification task requires the two classes to be separated as far as possible: $l(y, f) = l(-yf)$, while the regression tasks need predictions as close as possible to the target: $l(y, f) = \frac{1}{2}(y - f)^2$.

Each boosting round r ($1 \leq r \leq R$) selects a new weak learner $g_r : \mathbf{\Omega} \to \mathbb{R}$. This is added to the previously selected weak learners to obtain a better approximation of f: $f = \sum_{s \leq r} g_s$. In the most general case the new weak learner is selected to minimize the current cumulative loss:

$$g_r = \arg\min_{g \in \chi} L(f + g). \tag{2.1}$$

where χ represent the functional space of the weak learners. Finally, the **prediction** consists of evaluating the functional f on an unseen sample $x \in \mathbf{\Omega}$.

2.1.2 AdaBoost

AdaBoost (Freund, 1995) is the first boosting algorithm that combined weak learners slightly better than random into arbitrarily accurate strong learners. The algorithm was originally proposed for binary classification, with weak learners restricted to the functional space of $g : \mathbf{\Omega} \to \{-1, +1\}$. The

targets are also restricted to $y_n \in \{-1, +1\}$. This version is sometimes called **Discrete AdaBoost** (Friedman et al., 2000), because the weak learners are discrete.

Algorithm 1 The boosting algorithm AdaBoost.

Given: $\{(\mathbf{x}_n, y_n)_{n=1:N}\} \in (\mathbf{\Omega} \times \mathbb{R})^N$, where $y_n \in \{-1, +1\}$
Initialize distribution: $D_1(n) = 1/N$
for $r = 1$ **to** $r \leq R$ **do**
  Train weak learner: $g_r : \mathbf{\Omega} \to \{-1, +1\}$ using distribution D_r (Eq. 2.1)
  Choose: $\alpha_r = \frac{1}{2}\, log(\frac{1-\epsilon_r}{\epsilon_r}) > 0$
    where $\epsilon_r = \sum_n D_r(n)\ \mathbb{1}(g_r(\mathbf{x}_n) \neq y_n)$
  Update: $D_{r+1}(n) = \frac{D_r(n) \exp(-\alpha_r y_n g_r(\mathbf{x}_n))}{Z_r}$
    where Z_r is chosen such that D_{r+1} remains a distribution
end for
return $f = sign(\sum_{r \leq R} \alpha_r g_r)$

AdaBoost is detailed in Algorithm 1. The algorithm maintains and updates a normalization distribution D_r that associates a weight to each sample, proportional to its importance for the boosting round r. Initially, each sample has the same weight. The new learner g_r is selected to minimize the weighted mis-classification error $\sum_n D_r(n)\ \mathbb{1}(g_r(\mathbf{x}_n) \neq y_n)$, where $\mathbb{1}$ is the Kronecker delta function. Then it is scaled with α_r, which corresponds to the optimal line-search step in the direction of g_r, while minimizing the exponential loss $L(f) = \sum_n \exp(-y_n f(\mathbf{x}_n))$. More details are provided in the next section.

Once g_r is selected, the weights are updated based on its performance. If the weak learner classifies correctly a particular sample n, then the edge must be positive ($y_n g_r(\mathbf{x}_n) > 0$) and its weight $D_{r+1}(n)$ is decreased for the next round. Otherwise, the weight is increased. Thus, the new weak learners are trained to correct mistakes made by the current model. Each boosting round concentrates the weight distribution D to increasingly harder samples.

The AdaBoost algorithm is clearly independent of the weak learner's type. Boosted weak learners are usually decision stumps (Viola and Jones, 2001) or decision trees (Friedman et al., 2000). The boosted models can also be further combined in so called **cascades**. The most famous example of such a cascade was proposed by Viola and Jones (Viola and Jones, 2001, 2002). Their system was the first real-time face detector. The speed was achieved by combining two key ideas. The first idea is to use Haar-like features, that can be computed at any location and scale in constant time using integral images. The second idea is to classify a sample using a cascade of boosted decision stumps

of increasing complexity. The evaluation is performed by pruning the response of each boosted classifier, also called a **stage**, with a threshold. If the response is above the threshold, then the sample is rejected as background; otherwise, the sample is evaluated at the next (more complex) stage. Thus background samples, which account for the vast majority of evaluations, are quickly evaluated and rejected and the system spends more computation only near the face regions.

There are several **extensions** proposed to the AdaBoost algorithm, mostly using different loss functions. The original paper from (Freund, 1995) proposed three other boosting algorithms to solve multi-class classification (**AdaBoost.M1**, **AdaBoost.M2**) and regression (**AdaBoost.R**) problems. These algorithms are motivated from the perspective of achieving arbitrarily small training errors in logarithmic number of rounds. However, the theoretical results are valid for just some particular loss functions and weak learners. This limits the applicability of these boosting algorithms.

Other extensions to AdaBoost consist for example of pruning the weak learners that achieve high error rates, such as FloatBoost (Li et al., 2002), and of balancing the testing time and accuracy, such as WaldBoost (Sochman and Matas, 2005).

2.1.3 Boosting as functional gradient descent

A general motivation for boosting was proposed by several researchers (Friedman et al., 2000; Friedman, 2001; Duffy and Helmbold, 2002; Mason et al., 1999b). The key idea is to consider boosting as a greedy gradient descent performed in the functional space of the weak learners. Each boosting round is thus a gradient step towards minimizing a loss, usually convex, that matches the predictions of the strong learner $f(x)$ with the targets y. There are two steps performed at each boosting round:

1. The **selection** of the new weak learner (g_r) that locally decreases the loss the fastest:

$$g_r = \arg\min_g L(f + \epsilon g), \tag{2.2}$$

given a small variation ϵ. This corresponds to optimizing a local Taylor expansion of the loss (in the functional space of weak learners). The first order approximation gives gradient descent-like algorithms, while second order approximation gives Newton step-like algorithms.

2. The **scaling** of the selected weak learner g_r with the factor α_r. The strong learner is then updated as: $f \leftarrow f + \alpha_r g_r$. The scaling factor is required because the weak learner is only a local approximation of the loss. There are multiple possibilities to choose α_r, with the most commonly used being the line-search method:

$$\alpha_r = \arg\min_{\alpha} L(f + \alpha g_r). \tag{2.3}$$

However, fixed or decreasing steps are also a valid option. In these cases, the scaling factor can be interpreted as **learning rate**.

Formulating boosting as greedy gradient descent has the advantage that it can be used to solve both classification and regression problems in addition to univariate and multivariate problems. Most boosting algorithms can be derived from this interpretation (Masnadi-Shirazi, 2010). For example AdaBoost (Freund, 1995), Gentle AdaBoost (Friedman et al., 2000) or LogitBoost (Friedman et al., 2000) are instances that use particular loss functions or weak learners.

It is important to state that the weak learner and the scaling factor can be computed simultaneously or analytically in some cases (e.g. for AdaBoost), but this is not possible without prior knowledge on the loss and the weak learner type. There has also been work where each boosting round is complemented with an additional step that jointly re-optimized all the weak learners. This implies that previous weak learners (relative to the current round) are not fixed anymore. Most relevant to this idea is the ShareBoost algorithm (Shalev-Shwartz et al., 2011). There is some other related work on regularizing the loss (Xiang et al., 2009; Culp et al., 2010; Shivaswamy and Jebara, 2010, 2011), boosting structured weak learners (Fei and Huan, 2010) and decision trees (Friedman, 2001), and optimally sharing features for multi-class problems (Torralba et al., 2007).

Next we detail several important boosting algorithms relevant to, and that inspired, our proposed framework. All these algorithms are related to one another and use the two gradient steps describe above.

1. **GradientBoost** chooses the optimal weak learner as the one that aligns the best with the

local loss gradient (Friedman, 2001). The selection step is:

$$g_r = \arg\min_{g,\epsilon} \sum_n \left[\frac{\partial l(y_n, p)}{\partial p} \mid_{p=f(\mathbf{x}_n)} - \epsilon g(\mathbf{x}_n) \right]^2, \tag{2.4}$$

and the scaling step consists of line-search. The original paper describes several applications to different loss functions and makes use of decision trees as weak learners. The authors also introduce a **shrinkage** factor $\nu \in (0, 1]$. This changes the update of the strong learner to: $f \leftarrow f + \nu \alpha_r g_r$. The shrinkage factor controls the rate of the loss decrease and thus the speed of learning and implicitly of over-fitting. Thus shrinkage acts as a regularization method.

2. **AnyBoost** selects the optimal weak learner to point in the direction of the steepest descent (Mason et al., 2000). The optimal direction is given by the negative functional loss gradient. Generally, the space of weak learners χ may not contain one pointing exactly in the optimal direction. Thus a relaxation is needed that simply aligns the best new weak learner to the functional gradient:

$$g_r = \arg\max_g \; - \langle \nabla L(f), g \rangle. \tag{2.5}$$

Here ∇ is the gradient operator and $\langle \cdot, \cdot \rangle$ is the scalar product operator. No restriction is enforced on the scaling factor. The authors use a small constant, but they also suggest line-search as an alternative. The boosting rounds are stopped earlier if no weak learner can be found to point in the downhill direction of the loss function. Clearly, the AnyBoost formulation is closely related to GradientBoost as both algorithms use the same basic idea.

3. More recently, the **TaylorBoost** algorithm was introduced (Masnadi-Shirazi, 2010). Each boosting round is explicitly interpreted as optimizing the loss using a local Taylor expansion of order one or two. Let $\delta L(f, g) = \langle \nabla L(f), g \rangle$ and $\delta^2 L(f, g) = \langle \nabla^2 L(f), g^2 \rangle$ be the local variation of L, at point f along direction g. Then the selection step is shown to reduce to:

$$g_r = \arg\min_g \langle \nabla L(f), g \rangle \quad (GradBoost), \tag{2.6}$$

$$g_r = \arg\min_g \; -\frac{\langle \nabla L(f), g \rangle^2}{\langle \nabla^2 L(f), g^2 \rangle} \quad (QuadBoost). \tag{2.7}$$

The first order algorithm, named **GradBoost**, reduces to AnyBoost and GradientBoost. The authors show that AdaBoost is a particular instance of GradBoost. The second order algorithm, named **QuadBoost**, is a generalization of Newton step-based boosting algorithms such as LogitBoost. Both GradBoost and QuadBoost perform a line-search as the scaling step.

2.2 Local Binary Patterns

In this thesis we shall exclusively use local binary patterns (LBP) and their variations for all face processing tasks. This is because they are efficient to compute and have been proven to perform well for various challenging computer vision applications due to their robustness to illumination variation. Originally proposed for texture classification, the LBP family of features has successfully been applied to other problems such as face and object detection (Froba and Ernst, 2004; Zhang et al., 2007; Trefny and Matas, 2010), facial feature localization (Keomany, 2006) and face recognition (Liao et al., 2007; Tan and Triggs, 2010).

The family of LBP consists of features computed in a local, usually circular, neighborhood. The feature value is a binary code, where each bit is set depending on the comparison result between neighboring pixel values and the center pixel value (Ojala and Pietikainen, 2002).

Originally the LBP codes were proposed for 3×3 regions and thus had 2^8 distinct binary values. They were limited to small regions and could only capture local micro-textures. Thus the LBP was extended to $LBP_{(P,R)}$ where P is the number of points evenly spaced at radius R (expressed in pixels) from the center pixel. Some of the neighboring points require interpolation (usually linear) because their coordinates may not fall on pixel locations. The number of distinct values is increased to 2^P, because there are P comparisons with the center pixel value. Using this formulation an extended LBP was proposed which captures information at multiple scales by varying the radius R and from multiple resolutions by varying the number of sampling points P. An exhaustive survey on some other LBP extensions was recently published (Huang et al., 2011).

More closely related to our work are the **multi-block** (MB) (Zhang et al., 2007; Trefny and

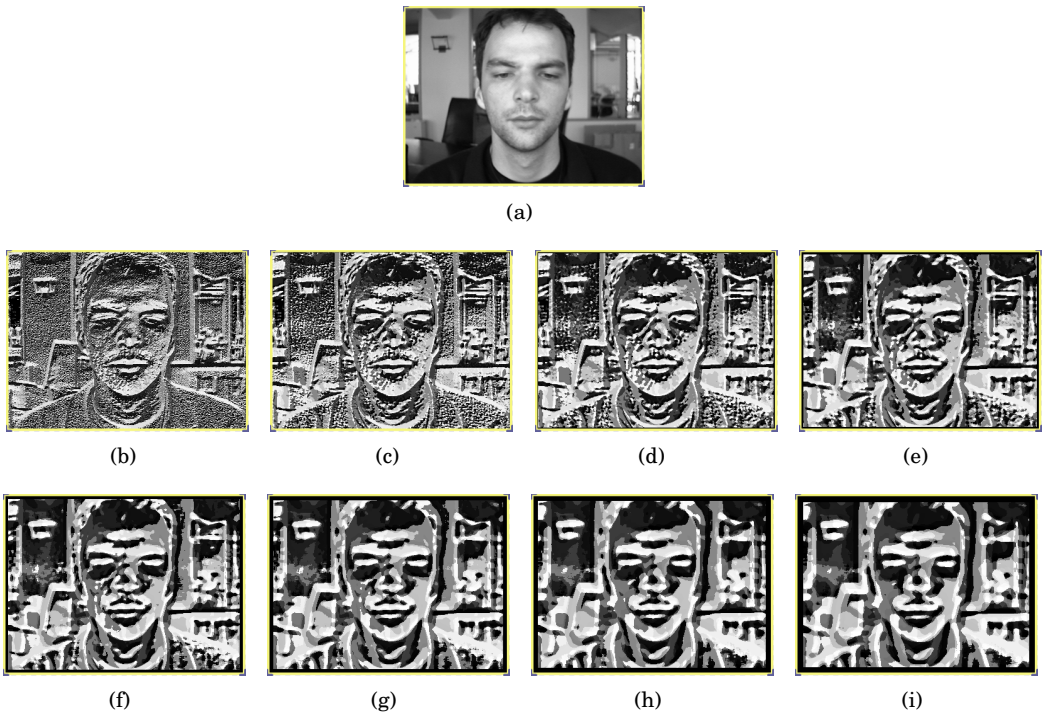


Figure 2.1. Illustration of the MB-LBP feature map feature maps for the original image (a). The multi-block patterns have cells of size 1×1 (b), 2×2 (c), 3×3 (d), 4×4 (e), 5×5 (f), 6×6 (g), 7×7 (h) and 8×8 (i) respectively.

Matas, 2010) extensions of LBP and the **Modified Census Transform** (MCT) (Froba and Ernst, 2004). The MB idea is similar to the $LBP_{(P,R)}$ extended set in that it captures information from multiple scales, but the neighboring is composed of 3×3 adjacent rectangular regions centered at the pixel of interest (see Fig. 2.3 (a)). The MCT codes are computed similarly to the LBP, with the difference that the average pixel values are used for comparison instead of the center pixel value in the LBP. Additionally, the MCT codes have an extra bit obtained by comparing the center pixel with the average. Typical MB-LBP feature maps are illustrated in Fig. 2.1. The coloring of the feature maps cannot be interpreted as pixel intensities; the coloring in this images is obtained by scaling the decimal value of the binary codes to the $[0, 256)$ range. It can be noticed that the coarseness of the local texture information is directly proportional with the size the cells: the smaller the multi-block cell, the finer the texture. However, the finest multi-block features are also less robust to noise or to various artifacts.

More formally, the MB features are parametrized by the top-left coordinate (x, y) in the patch to process and the size $cx \times cy$ of the rectangular cells of the 3×3 neighbourhood. Various patterns at

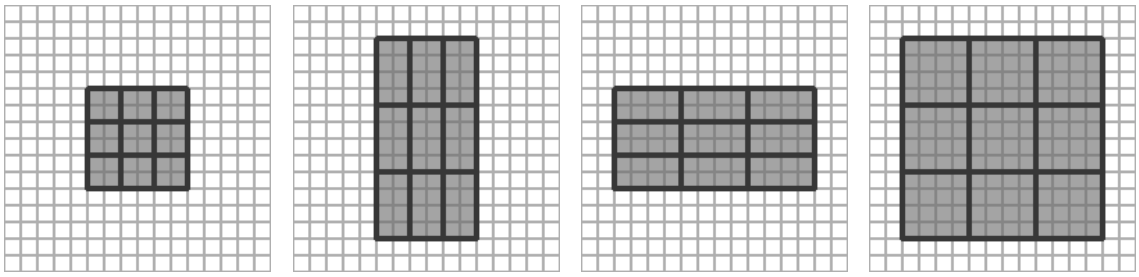


Figure 2.2. Various multi-block patterns of different cell sizes and at different locations illustrated on a generic 16×16 pixel grid.

multiple scales and aspect ratios can be obtained by varying these parameters (see Fig. 2.2 (b, c, d)). The richness of these patterns results in MB-LBP features outperforming both Haar-like features and LBP codes for the face detection task (Zhang et al., 2007).

Multi-block features can be generated for both LBP and MCT and their extensions. To do this we define p_i as the average pixel intensity in the cell i using the indexing presented in Fig. 2.3 (a). We refer to the centered pixel p_8 as p_c. Let $\bar{p} = \frac{1}{9} \sum_{i=0:8} p_i$ be the average pixel intensity in the 3×3 region. Then the LBP and the MCT codes are computed as presented in Table 2.1.

Recently, the transition LBP (tLBP) and the direction codes LBP (dLBP) have been proposed (Trefny and Matas, 2010) as extensions to the LBP. Considering the center pixel (region) as the origin, then the tLBP feature encodes circular transitions, while the dLBP feature encodes transitions across all four major directions. The combination of multi-block LBP, tLBP, dLBP and modified LBP (mLBP) encoding is denoted as **extended set of local binary patterns (EMB-LBP)** (Trefny and Matas, 2010). These LBP variations are illustrated in Fig. 2.3 and defined in Table 2.1.

The EMB-LBP features retain the robustness to illumination variation of the original LBP. The mLBP are more robust to random noise than the LBP, because they are computed by comparing the cells against the average intensity $\bar{p}$. The multi-block binary patterns are efficient to compute for any position and scale using the integral image (Viola and Jones, 2001). Furthermore, it can be argued that they are more efficient than the related $LBP_{(P,R)}$ because no interpolation is needed. Overall most of the LBP-like features are efficient enough for real-time applications.

In this thesis we shall exclusively use the EMB-LBP features, motivated by both their computational efficiency and superior performance (compared with MB-LBP features) on the face detection and the gender recognition tasks (Trefny and Matas, 2010).

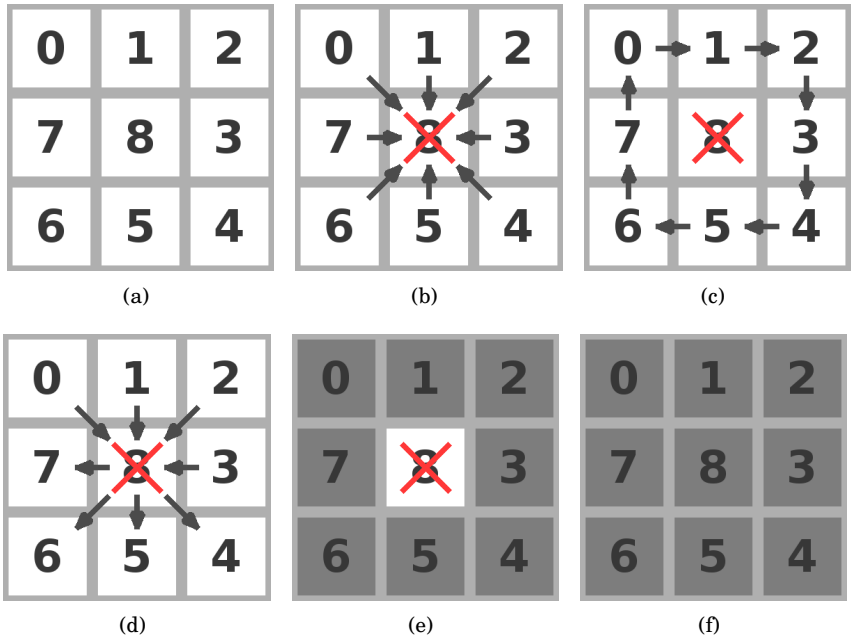


Figure 2.3. (a) Pixel indexing used for computing LBP and MCT patterns. (b, c, d, e, f) Illustration of the pixel (block) comparisons to obtain LBP, tLBP, dLBP, mLBP and MCT patterns respectively. The arrows denote the direction of the comparison. The grey cells are compared with the average within the 3×3 region, while the white cells are compared with the cell indicated by the arrow direction.

Name	**Operator**	**Range**
LBP	$\sum_{i=0:7} \mathbb{1}(p_i \geq p_c) \cdot 2^i$	$[0, 256)$
tLBP	$\sum_{i=0:7} \mathbb{1}(p_i \geq p_{(i+1) \mod 8}) \cdot 2^i$	$[0, 256)$
dLBP	$\sum_{i=0:3} \mathbb{1}((p_i - p_c)(p_{i+4} - p_c) \geq 0) \cdot 2^{2i} + \mathbb{1}(\lvert p_i - p_c \rvert \geq \lvert p_{i+4} - p_c \rvert) \cdot 2^{2i+1}$	$[0, 256)$
mLBP	$\sum_{i=0:7} \mathbb{1}(p_i \geq \bar{p}) \cdot 2^i$	$[0, 256)$
MCT	$\sum_{i=0:8} \mathbb{1}(p_i \geq \bar{p}) \cdot 2^i$	$[0, 512)$

Table 2.1. The $LBP_{(8,1)}$, the transition LBP (tLBP), the direction LBP (dLBP), the modified LBP (mLBP) and the MCT (multi-block) operators. The centered pixel is denoted as p_c. The LBP, mLBP, tLBP and dLBP feature set (denoted as EMB-LBP) shall be used throughout this thesis.

2.3 Summary

In this chapter, we have introduced boosting and local binary patterns, as the machine learning and the features used throughout this work. More specifically, the TaylorBoost algorithm and the MB-LBP features are the building blocks of the proposed framework. Let us summarize some important aspects that our work relies on:

1. TaylorBoost is a **generic** algorithm to combine weak learners (hypothesis) into strong learners, that achieve arbitrary accuracy on training samples. The implication is that it optimizes any smooth convex loss with any type of weak learner. Thus it can be used for a wide range of applications, both regression and classification. It is also **efficient** because the strong learner is constructed in a greedy fashion, by choosing one weak learner at each boosting round.

2. LBP features are **robust** to illumination variation and to random noise in the pixel values. On the other hand, the MB-LBP features integrates information from multiple scales. Both types of features are also **localized** and **efficient** to compute at any location and scale. These attributes makes them appropriate for **real-time vision applications**.

The next chapter presents the proposed framework that extends and combines these ideas. Firstly, we shall extend TaylorBoost to multivariate problems. The new algorithm is detailed for generic empirical expectation (cumulative) losses and variational losses. The variational formulation thus generalizes recent work on Expectation Bernstein Boosting (EBBoost) and Variational AdaBoost (VadaBoost). Second, we detail the proposed boosting algorithms to look-up-tables as weak learners. The optimization procedure is analyzed in terms of complexity and efficiency.

Chapter 3

A unified framework for boosting look-up tables

This chapter details the main contributions of this thesis which is to generalize and combine recent work on boosting. First, we introduce a generalization of the TaylorBoost algorithm to multivariate regression and classification problems. The multivariate TaylorBoost is analyzed for both expectation and variational loss formulations, the latter formulation being recently proposed by the EBBoost and VadaBoost algorithms (Shivaswamy and Jebara, 2010, 2011). Second, we detail the proposed boosting algorithms to LUTs as weak learners. We show that LUTs are efficient to boost (train) and to evaluate at test time.

3.1 Unified multivariate boosting framework

This section introduces a general boosting framework to learn multivariate classification and regression models. Besides some important advantages detailed below, this framework also extends and connects recent work on boosting, more precisely the TaylorBoost, EBBoost and VadaBoost algorithms.

3.1.1 Motivation

A general (unified) boosting framework ideally makes as few assumptions as possible on the loss function to optimize and the type of weak learners. The loss function is usually restricted to be analytic, smooth, convex and differentiable up to second order. This functional space contains a wide range of loss functions, that can be used to solve both univariate and multivariate classification and regression problems. An equally important property of such a boosting algorithm is to be computationally efficient. This implies that the weak learner selection step is efficient, because it is the most expensive step in boosting.

There are several advantages of the proposed boosting framework.

1. It is a unified learning algorithm that simplifies and enforces the comparison of different weak learners (and implicitly different features) and loss functions.

2. Its modular approach makes it easy to change components and to create and to experiment with new models.

3. Particular instances consist of well studied and successful boosting algorithms. Thus the boosting framework is consistent with previously established results for particular tasks. But its applicability can be extended to other tasks, for example multivariate regression problems.

The proposed approach extends previous work that proposed TaylorBoost (Masnadi-Shirazi, 2010). TaylorBoost optimizes efficiently any smooth loss function independent of the type of the weak learner. However, it is restricted to single-output (univariate) weak learners and to expectation loss functions that sum the loss values of the training samples.

The proposed approach circumvents these two limitations. First, we extend the first and second order TaylorBoost formulations, GradBoost and QuadBoost, to multiple-output (multivariate) weak learners. Second, we analyze a different general variational loss formulation that complements the original expectation formulation with the second order statistics of the loss value of the training samples. The multivariate TaylorBoost algorithm with a multivariate variational loss is a generalization of previous works that proposed the EBBoost and the VadaBoost algorithms (Shivaswamy and Jebara, 2010, 2011). The original work, on EBBoost and VadaBoost, discussed only the binary classification case, while we show in this thesis that it is possible to optimize a more general class

of multivariate losses.

3.1.2 TaylorBoost revised

Here, we present the original TaylorBoost algorithm in detail and introduced earlier in Chapter 2. The weak learners g and the strong learner f are univariate. Each boosting round constructs an improved model f with the goal of minimizing the loss $L(f)$ that measures the goodness of the predictions on the training samples. The loss $L(f)$ is chosen to be differentiable up to the second order.

The key idea is to choose the weak learners that locally decrease the loss the most. The loss is thus approximated locally (at the current estimation of the strong learner) using the Taylor expansion in the functional space of the weak learners:

$$L(f + \epsilon g) = L(f) + \epsilon \delta L(f, g) + \frac{\epsilon^2}{2} \delta^2 L(f, g) + \mathbf{O}(\epsilon^3), \tag{3.1}$$

where ϵ is the small variation in the direction of the weak learner g. The derivative of the functional $L(f)$ along the direction g is:

$$\delta L(f, g) = \frac{\partial L(f + \xi g)}{\partial \xi} \mid_{\xi=0}, \tag{3.2}$$

while similarly its curvature is defined as:

$$\delta^2 L(f, g) = \frac{\partial^2 L(f + \xi g)}{\partial \xi^2} \mid_{\xi=0} . \tag{3.3}$$

It is important to notice that this formulation makes no assumption on how the loss is decomposed over the training samples $\{(\mathbf{x}_n, y_n)_{n=1:N}\}$. Clearly $L(f)$ must be a composition of the loss values for each sample $l(y_n, f(\mathbf{x}_n))$. This is in the contrast with the original paper (Masnadi-Shirazi, 2010) where the loss is the empirical expectation of the loss values over the training samples: $L(f) = \sum_n l(y_n, f(\mathbf{x}_n))$.

Solving 3.1 for different orders of the loss approximation results in different boosting algorithms. GradBoost uses the first order (gradient) approximation, while QuadBoost uses the second order (quadratic) approximation. The two formulations are detailed in Algorithm 2. They differ only in

the weak learner selection step. GradBoost is a gradient descent method in the functional space of the weak learners, while QuadBoost is a Newton method. It can be shown that previously proposed boosting algorithms, like AdaBoost or LogitBoost, are particular instances of TaylorBoost (Masnadi-Shirazi, 2010). In some cases, depending on the loss function, the scaling factor α_r can be determined analytically and no line-search iterations are required.

Algorithm 2 The boosting algorithm TaylorBoost.

Given weak and strong learners: $g, f : \mathbf{\Omega} \rightarrow \mathbb{R}$
Given differentiable loss: $L(f)$
Initialize model: $f = 0$
for $r = 1$ **to** $r \leq R$ **do**
  Select weak learner:
    GradBoost: $g_r = \arg\min_g \delta L(f, g)$
    QuadBoost: $g_r = \arg\min_g -\frac{[\delta L(f,g)]^2}{\delta^2 L(f,g)}$
  Scale weak learner using line-search: $\alpha_r = \arg\min_\alpha L(f + \alpha g_r)$
  Update strong learner: $f \leftarrow f + \alpha_r g_r$
end for
return f

3.1.3 Multivariate TaylorBoost

TaylorBoost can be formulated as a multivariate optimization algorithm where the model predicts O values at a time. In this case, the targets, the weak and the strong learners become vectors of dimension O, while the base loss is changed similarly to compare multivariate targets and predictions. Before detailing these modifications, we arbitrarily denote vectors with $\mathbf{z}$ to distinguish from scalars z.

A specific output is referred with the index $1 \leq o \leq O$. Thus, the target of the sample n becomes $\mathbf{y}_n = (y_{o,n})_{1 \leq o \leq O}$. Similarly, the weak and strong responses become $\mathbf{g} = (g_o)_{1 \leq o \leq O}$ and $\mathbf{f} = (f_o)_{1 \leq o \leq O}$, respectively. The base loss changes to comparing vectors as: $l(\mathbf{y}, \mathbf{f}) : \mathbb{R}^O \times \mathbb{R}^O \rightarrow \mathbb{R}$. In this section, we make no assumption on the specific form of the base loss, except that it must be twice differentiable. Similarly, the overall loss $L(\mathbf{f})$ is an unknown mixture of the base loss values for the training samples.

The weak learners are scaled independently for each output when added to the strong learner. Thus, the strong learner update is performed element-wise: $(f_o \leftarrow f_o + \alpha_{o,r} g_{o,r})_{1 \leq o \leq O}$. This can be

written more compactly using the Hadamard operator: $\mathbf{f} \leftarrow \mathbf{f} + \boldsymbol{\alpha}_r \bullet \mathbf{g}_r$.

With these notations, we derive the multivariate TaylorBoost algorithm, denoted as **MTaylorBoost**. The weak learner is chosen to optimize the multivariate local Taylor expansion of the loss with O dimensions:

$$L(\mathbf{f} + \boldsymbol{\epsilon} \bullet \mathbf{g}) \approx L(\mathbf{f}) + \boldsymbol{\epsilon}^T \, \delta L(\mathbf{f}, \mathbf{g}) + \frac{1}{2} \boldsymbol{\epsilon}^T \, \delta^2 L(\mathbf{f}, \mathbf{g}) \, \boldsymbol{\epsilon}, \tag{3.4}$$

where the vector $\boldsymbol{\epsilon}$ is a small variation along direction $\mathbf{g}$. The gradient $\delta L(\mathbf{f}, \mathbf{g})$ is an O-dimensional vector, while the Hessian $\delta^2 L(\mathbf{f})$ is an $O \times O$ matrix. These functional derivatives are defined as:

$$\delta_o L(\mathbf{f}, \mathbf{g}) = \frac{\partial L(\mathbf{f} + (\xi \; \mathbb{1}_{-;o}) \bullet \mathbf{g})}{\partial \xi} \; |_{\xi=0}, \tag{3.5}$$

and:

$$\delta^2_{o_1,o_2} L(\mathbf{f}, \mathbf{g}) = \frac{\partial^2 L(f + (\xi_1 \; \mathbb{1}_{-;o_1}) \bullet \mathbf{g} + (\xi_2 \; \mathbb{1}_{-;o_2}) \bullet \mathbf{g})}{\partial \xi_1 \partial \xi_2} \; |_{\xi_1=\xi_2=0}, \tag{3.6}$$

respectively. The o component of the gradient quantifies the loss variation, at the limit, when the output o of the strong learner $\mathbf{f}$ is translated along the direction of the weak learner $\mathbf{g}$. Similarly, the (o_1, o_2) measures the loss variation, at the limit, when the two outputs o_1 and o_2 are modified simultaneously. The formulations make use of the notation $\mathbb{1}_{-;o}$ to denote a vector that has a unit response for the dimension o and zero for the other dimensions. This corresponds to the Kronecker delta function with two parameters:

$$\mathbb{1}_{p;o} = \delta_{p,o} = \begin{cases} 1, & \text{if } p = o \\ 0, & \text{if } p \neq o. \end{cases} \tag{3.7}$$

Depending on the order of the loss approximation, we obtain multivariate versions of GradBoost and QuadBoost denoted as **MGradBoost** and **MQuadBoost** respectively. These two multivariate boosting algorithms reduce **exactly** to the associated TaylorBoost formulation of the same order for single-output (univariate) problems. The only difference between MGradBoost and MQuadBoost is the weak learner selection step, similar to the univariate case.

MGradBoost

The MGradBoost algorithm chooses the weak learner as:

$$(\boldsymbol{\epsilon}^*, \mathbf{g}_r) = \arg\min_{\boldsymbol{\epsilon}, \mathbf{g}} \boldsymbol{\epsilon}^T \ \delta L(\mathbf{f}, \mathbf{g}), \tag{3.8}$$

to minimize the first order approximation of the loss. This an undefined optimization procedure, because the step $\boldsymbol{\epsilon}$ can be scaled to achieve arbitrarily low values. We thus fix the step to be unit for each output. Then the optimization procedure becomes:

$$\mathbf{g}_r = \arg\min_{\mathbf{g}} \sum_o \delta_o L(\mathbf{f}, \mathbf{g}). \tag{3.9}$$

This formulation can be interpreted as choosing the weak learner that has the highest loss decrease, summed over all the outputs. This is clearly a generalization of GradBoost (see Section 2), because the sum is removed in the case of single output problems ($O = 1$).

MQuadBoost

The MQuadBoost algorithm chooses the weak learner to minimize the second order approximation of the loss:

$$(\boldsymbol{\epsilon}^*, \mathbf{g}_r) = \arg\min_{\boldsymbol{\epsilon}, \mathbf{g}} \boldsymbol{\epsilon}^T \ \delta L(\mathbf{f}, \mathbf{g}) + \frac{1}{2} \boldsymbol{\epsilon}^T \ \delta^2 L(\mathbf{f}, \mathbf{g}) \ \boldsymbol{\epsilon}. \tag{3.10}$$

This optimization problem is simplified by eliminating the parameter $\boldsymbol{\epsilon}$. The optimal solution is achieved when the derivative, with respect to $\boldsymbol{\epsilon}$, of the optimization criteria vanishes:

$$\delta L(\mathbf{f}, \mathbf{g}) + \delta^2 L(\mathbf{f}, \mathbf{g}) \ \boldsymbol{\epsilon}^* = 0. \tag{3.11}$$

The optimal step is then:

$$\boldsymbol{\epsilon}^* = - \left[\delta^2 L(\mathbf{f}, \mathbf{g})\right]^{-1} \ \delta L(\mathbf{f}, \mathbf{g}). \tag{3.12}$$

Then 3.10 reduces to an optimization problem with a single variable:

$$\mathbf{g}_r = \arg\min_{\mathbf{g}} \; -\delta L(\mathbf{f},\mathbf{g})^T \; \left[\delta^2 L(\mathbf{f},\mathbf{g})\right]^{-1} \; \delta L(\mathbf{f},\mathbf{g}). \tag{3.13}$$

For the univariate case, the Hessian matrix reduces to the scalar $\delta^2 L(f,g)$, while the gradient is the scalar $\delta L(f,g)$. In this case equation 3.13 reduces exactly to the QuadBoost formulation. However, in the general multivariate case the Hessian matrix can be inefficient to invert for each boosting round, unless it can be performed analytically for some particular loss functions and weak learners.

MTaylorBoost

Both the first and second order weak learner selection procedures are followed by a multivariate line-search step of the form:

$$\boldsymbol{\alpha}_r = \arg\min_{\boldsymbol{\alpha}} L(\mathbf{f} + \boldsymbol{\alpha} \bullet \mathbf{g}_r), \tag{3.14}$$

where the vector $\boldsymbol{\alpha}_r$ adjusts each output of the selected weak learner to decrease the loss the most. Then, the strong learner is updated to: $\mathbf{f} \leftarrow \mathbf{f} + \boldsymbol{\alpha}_r \bullet \mathbf{g}_r$.

If no analytic solution can be computed for the optimal line-search steps, a gradient descent method would be typically used. These methods compute at each iteration the gradient of the criteria to optimize, with respect to its free parameters - $\boldsymbol{\alpha}$ in our case. It can be shown that this gradient is: $\delta L(\mathbf{f} + \boldsymbol{\alpha} \bullet \mathbf{g}, \mathbf{g})$. This gradient has the same formulation as the one used for the weak learner selection step, but is evaluated at the current estimation of the scaling factor $\boldsymbol{\alpha}$. Thus, the functional gradient formulation can be re-used once formulated.

Algorithm 3 summarizes the MTaylorBoost method. The algorithm is of little practical interest, unless the loss functions and the weak learners are specified. The following section presents two important overall loss functions and details the boosting steps for these losses. The next section further details the boosting algorithm for LUTs as weak learners. Finally, we describe the binary patterns used for boosting LUTs. Thus, we introduce an efficient boosting framework to be used for a wide range of multivariate classification and regression problems.

Algorithm 3 The boosting algorithm MTaylorBoost.

Given weak and strong learners: $\mathbf{g}, \mathbf{f} : \mathbf{\Omega} \rightarrow \mathbb{R}^O$
Given differentiable loss: $L(\mathbf{f})$
Initialize model: $\mathbf{f} = \mathbf{0}$
for $r = 1$ **to** $r \leq R$ **do**
    Select weak learner:
        MGradBoost: $\mathbf{g}_r = \arg\min_{\mathbf{g}} \sum_o \delta_o L(\mathbf{f}, \mathbf{g})$
        MQuadBoost: $\mathbf{g}_r = \arg\min_{\mathbf{g}} \; -\delta L(\mathbf{f}, \mathbf{g})^T \; \left[\delta^2 L(\mathbf{f}, \mathbf{g})\right]^{-1} \; \delta L(\mathbf{f}, \mathbf{g})$
    Scale weak learner using line-search: $\boldsymbol{\alpha}_r = \arg\min_{\boldsymbol{\alpha}} L(\mathbf{f} + \boldsymbol{\alpha} \bullet \mathbf{g}_r)$
    Update strong learner: $\mathbf{f} \leftarrow \mathbf{f} + \boldsymbol{\alpha}_r \bullet \mathbf{g}_r$
end for
return $\mathbf{f}$

3.1.4 Overall loss functions

This section describes in more detail the overall loss function $L(\mathbf{f})$. We refer to this loss as an *overall loss* because it mixes the loss values of the training samples. More formally, let $l(\mathbf{y}, \mathbf{f}) : \mathbb{R}^O \times \mathbb{R}^O \rightarrow \mathbb{R}$ be an unspecified twice differentiable **base loss** function and $\{(\mathbf{x}_n, \mathbf{y}_n)_{n=1:N}\} \in (\mathbf{\Omega} \times \mathbb{R}^O)^N$ be a set of training samples. The base loss $l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n))$ evaluated for a particular sample $(\mathbf{x}_n, \mathbf{y}_n)$ measures the error produced by predicting $\mathbf{f}(\mathbf{x}_n)$ instead of $\mathbf{y}_n$.

In the most general case, the overall loss combines the error distributions over all of the training samples as:

$$L(\mathbf{f}) = L\left(\{l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n))\}_{n=1:N}\right). \tag{3.15}$$

We describe two overall losses that use first or second order statistics of the error distribution over the training samples. The first order (**expectation loss**) aims at minimizing the average error distribution, while the second order (**variational loss**) complements the expectation loss with a regularization term proportional with the empirical variance of the error distribution. We detail the multivariate derivatives for both losses in the form required by the MTaylorBoost algorithm.

To simplify the equations we introduce some useful notation. The loss values and first and second order derivatives for a sample n using the predictions $\mathbf{f}$ are denoted as:

$$l_n(\mathbf{f}) = l_n = l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n)), \tag{3.16}$$

$$l'_{o,n}(\mathbf{f}) = l'_{o,n} = \frac{\partial l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n) + (\xi \ \mathbb{1}_{-;o}))}{\partial \xi} \ |_{\xi=0}, \tag{3.17}$$

$$l''_{o_1,o_2,n}(\mathbf{f}) = l''_{o_1,o_2,n} = \frac{\partial^2 l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n) + (\xi_1 \ \mathbb{1}_{-;o_1}) + (\xi_2 \ \mathbb{1}_{-;o_2}))}{\partial \xi_1 \partial \xi_2} \ |_{\xi_1=\xi_2=0}, \tag{3.18}$$

respectively. We sometimes omit the predictions to unclutter equations, when it is clear from the context that $\mathbf{f}$ is the strong learner from the previous round.

Let us denote the loss distribution as $\mathbb{L} = \{(l_n)_{n=1:N}\}$. The empirical expectation and variance of the $\mathbb{L}$ are denoted as:

$$\hat{\mathbf{E}}[\mathbb{L}] = \frac{1}{N}\sum_n l_n, \tag{3.19}$$

and:

$$\hat{\mathbf{V}}[\mathbb{L}] = \hat{\mathbf{E}}\left[\left(\mathbb{L} - \hat{\mathbf{E}}[\mathbb{L}]\right)^2\right]. \tag{3.20}$$

Expectation loss

The expectation loss is chosen to penalize strong learners that do not perform well *on average*. This is probably the most widely used loss formulation by the machine learning community. More formally, the expectation loss is defined as:

$$\begin{aligned} L_E(\mathbf{f}) &= \sum_n l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n)) \\ &= \sum_n l_n \\ &\propto \hat{\mathbf{E}}[\mathbb{L}]. \end{aligned} \tag{3.21}$$

The derivatives required by MTaylorBoost are computed by summing over the samples the derivatives of the base loss. Following the chain rule, a component of the functional gradient is decomposed as:

$$\delta_o L_E(\mathbf{f}, \mathbf{g}) = \sum_n \frac{\partial l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n) + (\xi \ \mathbb{1}_{-;o}) \bullet \mathbf{g}(\mathbf{x}_n))}{\partial \xi} \mid_{\xi=0}$$
$$= \sum_n l'_{o,n} \ \mathbf{g}_o(\mathbf{x}_n). \tag{3.22}$$

This is the scalar product of the loss gradients and the responses of the weak learner over the training samples: $\delta_o L_E(\mathbf{f}, \mathbf{g}) = \langle \nabla_o L_E(\mathbf{f}), \mathbf{g}_o \rangle$.

The same procedure is applied to a component of the functional Hessian to obtain:

$$\delta^2_{o_1,o_2} L_E(\mathbf{f}, \mathbf{g}) = \sum_n \frac{\partial^2 l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n) + (\xi_1 \ \mathbb{1}_{-;o_1}) \bullet \mathbf{g}(\mathbf{x}_n) + (\xi_2 \ \mathbb{1}_{-;o_2}) \bullet \mathbf{g}(\mathbf{x}_n))}{\partial \xi_1 \partial \xi_2} \mid_{\xi_1=\xi_2=0}$$
$$= \sum_n l''_{o_1,o_2,n} \ \mathbf{g}_{o_1}(\mathbf{x}_n) \ \mathbf{g}_{o_2}(\mathbf{x}_n). \tag{3.23}$$

Similarly, this is the scalar product of the loss for the second order derivatives and the responses of the weak learners over the training samples: $\delta^2_{o_1,o_2} L_E(\mathbf{f}, \mathbf{g}) = \langle \nabla^2_{o_1,o_2} L_E(\mathbf{f}), \mathbf{g}_{o_1} \bullet \mathbf{g}_{o_2} \rangle$.

It can be noticed that both derivatives are scalar products between constants, based on the current strong learner, and the predictions of the new weak learner to add. This has practical implications, because these constant terms ($l'_{o,n}$ and l''_{o_1,o_2}) can be buffered before selecting the weak learner. Thus the optimization can be significantly sped up. However, this does not help for the MQuadBoost algorithm.

The MQuadBoost algorithm requires the inversion of the Hessian matrix which is generally expensive if the matrix presents no structure. Also, this algorithm depends on the unknown optimal weak learner. If the base loss l has a diagonal Hessian matrix, then the algorithm simplifies to a sum of independent QuadBoost problems which is more efficient to solve. More formally, the selection of the optimal weak learner reduces to:

$$\mathbf{g}_r = \arg\min_{\mathbf{g}} \ -\sum_o \frac{[\delta_o L_E(\mathbf{f}, \mathbf{g})]^2}{\delta^2_{o,o} L_E(\mathbf{f}, \mathbf{g})}. \tag{3.24}$$

Variational loss

The motivation for using a variational loss is to penalize not only wrong predictions, but also large variations of prediction accuracy. A smaller variation of the prediction errors, for the training samples, is intuitively related to a higher confidence in predictions. Thus, it is expected that a

model trained with such a loss will generalize better on unseen data. This intuition is supported by theoretical results that estimate the bounds of the generalization error. Some of these results are described in recent work (Shivaswamy and Jebara, 2010, 2011).

More formally, the variational loss trades-off the empirical loss expectation with the empirical loss variance:

$$L_V(\mathbf{f}) = \beta \ \left(\hat{\mathbf{E}}[\mathbb{L}]\right)^2 + \gamma \ \hat{\mathbf{V}}[\mathbb{L}]. \tag{3.25}$$

Following the work of (Shivaswamy and Jebara, 2010, 2011), this loss can be reformulated as:

$$\begin{aligned} L_V(\mathbf{f}) &= \left[\sum_n l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n))\right]^2 + \lambda \sum_{n>m} \left[l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n)) - l(\mathbf{y}_m, \mathbf{f}(\mathbf{x}_m))\right]^2 \\ &= \left(\sum_n l_n\right)^2 + \lambda \sum_{n>m} (l_n - l_m)^2, \end{aligned} \tag{3.26}$$

with the trade-off parameter $\lambda \geq 0$. Setting the trade-off parameter to zero, reverts the variational formulation to the associated expectation formulation for the same base loss function. A sufficiently large λ value may bias the model towards predictions that are almost constant for any sample, because constant distributions have minimum variance. In practice, the optimal trade-off parameter is tuned on a distinct validation dataset.

There has been recent work on boosting that makes use of specific variational losses. The EBBoost (Empirical Bernstein Boosting) (Shivaswamy and Jebara, 2010) and the VadaBost (Variational AdaBoost) (Shivaswamy and Jebara, 2011) were proposed to solve binary classification problems and they both make use of the exponential base loss: $l(y, f) = \exp(-yf)$. Both algorithms are reduced to an AdaBoost formulation of the same complexity, but with slightly modified weight distributions and scaling factors.

In this thesis, we use the more general variational loss of Eq. 3.26 within the MTaylorBoost framework. We derive the gradient and the Hessian components similarly to the expectation loss case using the chain rule. But first, we rewrite Eq. 3.26 to simplify this computation:

$$
\begin{aligned}
L_V(\mathbf{f}) &= \left(\sum_n l_n\right)^2 + \lambda \sum_{n>m} (l_n - l_m)^2 \\
&= \sum_n l_n^2 + 2\sum_{n>m} l_n l_m + \lambda \sum_{n>m} (l_n^2 + l_m^2 - 2 l_n l_m) \\
&= \left[\lambda \sum_{n>m} (l_n^2 + l_m^2) + \lambda \sum_n l_n^2\right] + \left[(1-\lambda)\left(\sum_n l_n^2 + 2\sum_{n>m} l_n l_m\right)\right] \\
&= \lambda \frac{1}{2} \sum_{n,m} \left(l_n^2 + l_m^2\right) + (1-\lambda)\left(\sum_n l_n\right)^2 \\
&= \lambda N \left(\sum_n l_n^2\right) + (1-\lambda)\left(\sum_n l_n\right)^2 .
\end{aligned} \tag{3.27}
$$

Then the gradient becomes:

$$
\delta_o L_V(\mathbf{f}, \mathbf{g}) = 2 \sum_n \left[\lambda N l_n + (1-\lambda) L_E(\mathbf{f})\right]\ l'_{o,n}\ \mathbf{g}_o(\mathbf{x}_n). \tag{3.28}
$$

It can be noticed that the variational gradient is formulated similar to the expectation gradient as a scalar product of some constants and the weak learner responses: $\delta_o L_V(\mathbf{f}, \mathbf{g}) = \langle \nabla_o L_V(\mathbf{f}), \mathbf{g}_o \rangle$. Thus, MGradBoost selects the optimal weak learner with the same complexity, the only difference being the actual buffered constants.

A similar result can be obtained for the Hessian. For the variational loss the Hessian becomes:

$$
\begin{aligned}
\delta^2_{o_1,o_2} L_V(\mathbf{f}, \mathbf{g}) = {} & 2 \sum_n \left[\lambda N l_n + (1-\lambda) L_E(\mathbf{f})\right]\ l''_{o_1,o_2,n}\ \mathbf{g}_{o_1}(\mathbf{x}_n)\ \mathbf{g}_{o_2}(\mathbf{x}_n) \\
& + 2 \sum_n \left[\lambda N l'_{o_1,n} l'_{o_2,n}\right]\ \mathbf{g}_{o_1}(\mathbf{x}_n)\ \mathbf{g}_{o_2}(\mathbf{x}_n) \\
& + 2(1-\lambda) \left[\sum_n l'_{o_1,n}\ \mathbf{g}_{o_1}(\mathbf{x}_n)\right] \left[\sum_n l'_{o_2,n}\ \mathbf{g}_{o_2}(\mathbf{x}_n)\right].
\end{aligned} \tag{3.29}
$$

A diagonal Hessian for an expectation loss does not translate to a diagonal Hessian for the associated variational loss. This is because of the second order terms, that do not depend on the second order derivative of the base loss. Thus, the MQuadBoost algorithm does not simplify for L_V to a sum of univariate algorithms, even if it simplifies for the associated L_E. Usually an iterative optimization procedure, with $\boldsymbol{\epsilon}$ and $\mathbf{g}$ as parameters, must be used to select the optimal weak learner for the MQuadBoost case. This is in constrast with the MGradBoost algorithm, where an analytical

solution can be found. Thus the first order algorithm is much more efficient than the second order one for a generic loss.

Expectation and Variational MGradBoost

We detail the weak selection and the line-search steps of MGradBoost. To simplify notations, we denote $L'_{o,n}(\mathbf{f})$ as the gradient for the output o and the sample n using the $\mathbf{f}$ strong learner. This gradient is reduced to:

$$L'_{E,o,n}(\mathbf{f}) = l'_{o,n}(\mathbf{f}), \tag{3.30}$$

$$L'_{V,o,n}(\mathbf{f}) = 2\left[\lambda N l_n(\mathbf{f}) + (1-\lambda) L_E(\mathbf{f})\right] l'_{o,n}(\mathbf{f}), \tag{3.31}$$

for the expectation and the variations formulations respectively. We have showed in the previous section that the accumulated loss gradient is decomposed as the scalar product between some constants (with respect to the weak learner) and the weak learner predictions, for both expectation and variational formulations. Then the weak learner is selected using:

$$\mathbf{g}_r = \arg\min_{\mathbf{g}} \sum_{o,n} L'_{o,n}(\mathbf{f})\ \mathbf{g}_o(\mathbf{x}_n) \tag{3.32}$$

Algorithm 4 details the first order weak learner selection step for the expectation and the variational loss formulations. It can be noticed that in both cases this reduces to optimizing a linear equation between different constants (proportional with the loss gradients) and the same weak learner predictions. Thus, both formulations have the same complexity for given weak learners.

3.2 Boosting look-up-tables

In this work we have chosen to boost LUTs, also named multi-branch decision trees (Zhang et al., 2007). These weak learners present several advantages. First, the boosting algorithm MGradBoost has a simple and efficient formulation for this type of weak learner for both expectation and variational losses. Second, the LUT can be used with a wide range of discrete positive features: both codes (e.g. LBP, MCT (Froba and Ernst, 2004)) and features that lie in the Euclidean space (e.g. greyscale, histograms). And finally, the LUT has been successfully boosted for state-of-the-art face

Algorithm 4 The boosting algorithm MGradBoost detailed for the expectation and the variational loss formulations.

Given weak and strong learners: $\mathbf{g}, \mathbf{f} : \mathbf{\Omega} \rightarrow \mathbb{R}^O$
Given differentiable loss: $L(\mathbf{f})$
Initialize model: $\mathbf{f} = \mathbf{0}$
for $r = 1$ **to** $r \leq R$ **do**
  Select weak learner: $\mathbf{g}_r = \arg\min_{\mathbf{g}} \sum_{o,n} L'_{o,n}(\mathbf{f})\ \mathbf{g}_o(\mathbf{x}_n)$
    Expectation: $L'_{o,n}(\mathbf{f}) = l'_{o,n}(\mathbf{f})$
    Variational: $L'_{o,n}(\mathbf{f}) = [\lambda N l_n(\mathbf{f}) + (1-\lambda) L_E(\mathbf{f})]\ l'_{o,n}(\mathbf{f})$
  Scale weak learner using line-search: $\boldsymbol{\alpha}_r = \arg\min_{\boldsymbol{\alpha}} L(\mathbf{f} + \boldsymbol{\alpha} \bullet \mathbf{g}_r)$
  Update strong learner: $\mathbf{f} \leftarrow \mathbf{f} + \boldsymbol{\alpha}_r \bullet \mathbf{g}_r$
end for
return f

detection (Froba and Ernst, 2004; Zhang et al., 2007). We consider that these weak learners can be used for other tasks too, for example regression.

The LUT is parametrized by a feature index (a particular dimension in the feature space) and a set of fixed outputs, one for each distinct feature value. More formally, the weak learner g is computed for a sample $\mathbf{x}$ and a feature d with:

$$g(\mathbf{x}) = g(\mathbf{x}; d, \mathbf{a}) = \mathbf{a}[u = \mathbf{x}_d], \tag{3.33}$$

where $\mathbf{a}$ is the LUT with U entries $\mathbf{a}_u$. The feature value $\mathbf{x}_d$ is used as an index u in the look-up table. The number of entries U is actually the number of distinct values the features can have that for simplicity we consider to be in the range $[0, U)$.

The extension to the multivariate case is performed by concatenating LUTs for each output:

$$\mathbf{g}(\mathbf{x}) = \mathbf{g}(\mathbf{x}; \mathbf{d}, \mathbf{A}) = \{\mathbf{A}_o[u = \mathbf{x}_{\mathbf{d}_o}]\}_{1 \leq o \leq O}, \tag{3.34}$$

where each output $\mathbf{g}_o$ has a different set of entries $\mathbf{A}_o$ of the same size U and possibly a different feature $\mathbf{d}_o$. We refer to $\mathbf{A}_{o,u}$ and $\mathbf{A}_o[u]$ as the o^{th} output indexed by the feature value u in the LUT entries $\mathbf{A}_o$.

The goal of the boosting algorithm is to compute the optimum features $\mathbf{d}$ and the optimal entries $\mathbf{A}_{o,u}$. Boosting LUTs thus performs **feature selection**, because the selected weak learners are each parametrized by a single feature for each output. We shall investigate two situations. The first

consists of **sharing** a single feature ($\mathbf{d}_o = d$) for all outputs, which we denote as **MGradBoost-Shared**. The second consists of selecting the features **independently** for each output, which we denote as **MGradBoost-Indep**. The sharing formulation results in fewer features to evaluate at test time, thus it provides simpler and faster models. However, the shared models may not have the representation power of the independent formulations.

3.2.1 Weak learner selection step

Here we detail the weak learner selection step of the MGradBoost algorithm for the two cases of sharing features between outputs. This step reduces for MGradBoost-Shared to:

$$\begin{aligned}
\mathbf{g}_r &= \arg\min_{\mathbf{g}} \sum_{o,n} L'_{o,n}(\mathbf{f})\ \mathbf{g}_o(\mathbf{x}_n) \\
&= \arg\min_{d,\mathbf{A}} \sum_{o,n} L'_{o,n}(\mathbf{f})\ \mathbf{g}_o(\mathbf{x}_n; d, \mathbf{A}) \\
&= \arg\min_{d} \sum_{o} \left\{ \arg\min_{\mathbf{A}_o} \sum_{n} L'_{o,n}(\mathbf{f})\ \mathbf{g}_o(\mathbf{x}_n; d, \mathbf{A}_o) \right\} \\
&= \arg\min_{d} \sum_{o} \left\{ \arg\min_{\mathbf{A}_o} \sum_{u} \left[\sum_{n, \mathbf{x}_{n,d}=u} L'_{o,n}(\mathbf{f}) \right] \mathbf{A}_{o,u} \right\},
\end{aligned} \tag{3.35}$$

and for MGradBoost-Indep to:

$$\begin{aligned}
\mathbf{g}_r &= \arg\min_{\mathbf{g}} \sum_{o,n} L'_{o,n}(\mathbf{f})\ \mathbf{g}_o(\mathbf{x}_n) \\
&= \arg\min_{\mathbf{d},\mathbf{A}} \sum_{o,n} L'_{o,n}(\mathbf{f})\ \mathbf{g}_o(\mathbf{x}_n; \mathbf{d}, \mathbf{A}) \\
&= \sum_{o} \arg\min_{\mathbf{d}_o,\mathbf{A}_o} \sum_{n} L'_{o,n}(\mathbf{f})\ \mathbf{g}_o(\mathbf{x}_n; \mathbf{d}_o, \mathbf{A}_o) \\
&= \sum_{o} \left\{ \arg\min_{\mathbf{d}_o=d} \arg\min_{\mathbf{A}_o} \sum_{u} \left[\sum_{n, \mathbf{x}_{n,d}=u} L'_{o,n}(\mathbf{f}) \right] \mathbf{A}_{o,u} \right\},
\end{aligned} \tag{3.36}$$

respectively. We have used the fact that look-up-tables produce a constant output $\mathbf{A}_{o,u}$ for the subset of the samples that have the fixed feature d with the value u. Thus each look-up-table entry is selected independently of the others as:

$$\mathbf{A}_{o,u} = -\operatorname{sgn}\left[\sum_{n,\mathbf{x}_{n,\mathbf{d}_o}=u} L'_{o,n}(\mathbf{f})\right], \tag{3.37}$$

for a fixed feature $\mathbf{d}_o$ attributed to the output o. The optimal entry value is proportional to the sum of the loss gradients of the samples that fall in the associated bin u. This sum can be interpreted as the cumulated mis-predictions that the entry should decrease. Replacing the optimal look-up-table entry values in the initial equations, the optimal shared feature is:

$$d^* = \arg\min_d \; -\sum_{o,u} \left| \sum_{n,\mathbf{x}_{n,d}=u} L'_{o,n}(\mathbf{f}) \right|, \tag{3.38}$$

while the optimal independent feature is:

$$\mathbf{d}_o^* = \arg\min_d \; -\sum_{u} \left| \sum_{n,\mathbf{x}_{n,d}=u} L'_{o,n}(\mathbf{f}) \right|, \tag{3.39}$$

for the output o. It can be noticed that the criteria to choose the optimal shared feature is a sum over the criteria to choose the independent features. Once the feature(s) selection is performed, the optimal look-up-table entries are computed using Eq. 3.37 to obtain $\mathbf{A}^*_{o,u}$. Finally, the optimal weak learner is thus $\mathbf{g}_r = \mathbf{g}(;\mathbf{d}^*, \mathbf{A}^*)$.

3.2.2 Line-search step

The line-search step consists of computing the optimal scaling factors: $\boldsymbol{\alpha}_r = \arg\min_{\boldsymbol{\alpha}} L(\mathbf{f} + \boldsymbol{\alpha} \bullet \mathbf{g}_r)$. This optimization problem has O variables - one for each output, and it can be solved using off-the-shelf optimization algorithms. Such an algorithm requires iteratively the gradient of the optimization criteria in respect with each variable, until no improvement can be made. These gradients can be computed at each iteration using the chain rule as:

$$\begin{aligned} \frac{\partial L(\mathbf{f} + (\boldsymbol{\alpha} + \xi \; \mathbb{1}_{-;o}) \bullet \mathbf{g}_r)}{\partial \xi} \; |_{\xi=0} &= \delta_o L(\mathbf{f} + \boldsymbol{\alpha} \bullet \mathbf{g}_r, \mathbf{g}_r) \\ &= \sum_n L'_{o,n}(\mathbf{f} + \boldsymbol{\alpha} \bullet \mathbf{g}_r) \; \mathbf{g}_{o,r}(\mathbf{x}_n) \end{aligned} \tag{3.40}$$

where $\boldsymbol{\alpha}$ is the vector of current scaling factor. This is exactly the same formulation used for selecting the optimal weak learner, but at the translated strong learner $\mathbf{f} + \boldsymbol{\alpha} \bullet \mathbf{g}_r$ with the current

scaling factors. It can be noticed that the constants in the cross product are now the weak learner predictions $\mathbf{g}_{o,r}(\mathbf{x}_n)$ which do not depend on α. But the $L'_{o,n}(\mathbf{f}+\boldsymbol{\alpha}\bullet\mathbf{g}_r)$ terms need to be re-computed at each iteration. The line-search was performed using the C-library **libLBFGS** that implements the L-BFGS optimization algorithm (Nocedal and Jorge, 1989).

3.2.3 MGradBoost for boosting look-up-tables

We summarize the proposed boosting look-up-tables method in Algorithm 5. Each boosting round consists of three major steps. First, feature selection is performed by computing d^* or $\mathbf{d}^*_o$ (step 6 or step 7). Given that D features are available, this step has the complexity $\mathcal{O}(D \times O \times N)$ because it needs to sum the loss gradients for each sample and outputs. Second, the optimal entries are computed using step 8 with the complexity $\mathcal{O}(D \times O \times N)$. Finally, line-search is performed to compute the scaling factors using some iterative gradient descent algorithm. Each iteration involves summing similar loss gradients with complexity $\mathcal{O}(D \times O \times N)$ (see Eq. 3.40). Overall, the training complexity is thus $\mathcal{O}(R \times D \times O \times N)$ depending **linearly with the number of boosting rounds, features, outputs and training samples**.

Algorithm 5 The MGradBoost algorithm for boosting look-up-tables with shared features.

1: Given weak and strong learners: $\mathbf{g}, \mathbf{f} : \boldsymbol{\Omega} \rightarrow \mathbb{R}^O$
2: Given differentiable loss: $L(\mathbf{f})$
3: Initialize model: $\mathbf{f} = \mathbf{0}$
4: **for** $r = 1$ **to** $r \leq R$ **do**
5: Select weak learner: $\mathbf{g}_r = \mathbf{g}(;\mathbf{d}^*, \mathbf{A}^*)$, where:
6: **MGradBoost-Shared**: $d^* = \arg\min_d \; -\sum_{o,u} |\sum_{n,\mathbf{x}_{n,d}=u} L'_{o,n}(\mathbf{f})|$
7: **MGradBoost-Indep**: $\mathbf{d}^*_o = \arg\min_d \; -\sum_{u} |\sum_{n,\mathbf{x}_{n,d}=u} L'_{o,n}(\mathbf{f})|$
8: $\mathbf{A}_{o,u} = -\operatorname{sgn}[\sum_{n,\mathbf{x}_{n,\mathbf{d}^*_o}=u} L'_{o,n}(\mathbf{f})]$
9: Scale weak learner using line-search: $\boldsymbol{\alpha}_r = \arg\min_{\boldsymbol{\alpha}} L(\mathbf{f} + \boldsymbol{\alpha} \bullet \mathbf{g}_r)$
10: Update strong learner: $\mathbf{f} \leftarrow \mathbf{f} + \boldsymbol{\alpha}_r \bullet \mathbf{g}_r$
11: **end for**
12: **return** f

The evaluation of the boosted strong learner $\mathbf{f}$ on a sample $\mathbf{x}$ consists of summing the weak learner responses for each output: $\mathbf{f}_o(\mathbf{x}) = \sum_{r \leq R} \mathbf{g}_{o,r}(\mathbf{x}; \mathbf{d}_r, \mathbf{A}_r)$. The summation involves between R (MGradBoost-Shared) and $R \times O$ (MGradBoost-Indep) feature computations and $R \times O$ elementary LUT indexing operations and additions. Usually, the feature computation is the most time consum-

ing operation at test time, while the memory access and the additions are almost negligible. Thus we argue that LUTs are efficient to boost and to evaluate.

The analysis has shown that the feature selection is the most computationally intensive step. However, this step is easily parallelized by noticing that the criteria for each feature can be computed independently. The feature collection to evaluate can be thus split uniformly across multiple processing units. This makes the boosting algorithm **scalable** with the available resources.

3.3 Summary

This chapter introduced a generic boosting framework for multivariate classification and regression problems. The proposed MTaylorBoost algorithm is detailed for the expectation and the variational loss formulations. We show that boosting LUTs is efficient and scalable for both loss formulations and the first order MTaylorBoost algorithm. Overall, the proposed framework is geared towards building efficient models suitable for a wide range of real-time applications. The next chapter describes how to efficiently boost high resolution models using large pools of samples.

Chapter 4

Efficient boosting

This chapter presents the main contributions to boost models efficiently when the number of both features and samples is very high. Our proposed approach addresses the case of high resolution models and of very large training pools, that usually occur in face processing tasks. First, we introduce a coarse-to-fine feature selection method for multi-block LBPs. This method reduces the number of features of high resolution models to practical values by iteratively refining the boosted features. Second, we present a generic bootstrapping algorithm to sample representative training data from very large sample pools. This algorithm iteratively builds a better representation of the available training data.

4.1 Coarse-to-fine multi-block feature selection

Multi-block Local Binary Patterns (MB-LBP) are widely used for face detection (Froba and Ernst, 2004; Zhang et al., 2007; Trefny and Matas, 2010). This is because these features are efficient to compute, robust to illumination and to some degree to noise, while achieving good performance. The models usually have fairly low resolution (e.g. of 24×24 pixels). The number of features increases rapidly to millions for high resolution models, thus boosting such models proves unfeasible. However, high resolution models are often required in high precision tasks, for example to precisely locate particular facial features. To address this problem of efficiently boosting multi-block features for high resolution models we propose a coarse-to-fine feature selection (CTFFS) procedure.

	Model size		
	20×24	40×48	80×96
Number of features	19,100	355,700	6,112,700
Number of features / pixel	39.8	185.3	795.9

Table 4.1. The number of extended multi-block Local Binary Patterns (EMB-LBP) generated for various model sizes. The feature redundancy increases fast with the model resolution.

Multi-block patterns

First, we define a **multi-block pattern** as consisting of two components.

1. A **rectangular region** where features are extracted from, defined by its top-left (t, l) coordinates. This region is broken into $n_x \times n_y$ adjacent and rectangular cells (or blocks) of the same size in pixels $c_x \times c_y$. Then the size in pixels of region is $n_x c_x \times n_y c_y$. For example, the LBP features are patterns of 3×3 cells each consisting of one pixel, while the MB-LBP features are similarly patterns of 3×3 cells of variable size.

2. An **operator** that uses the average pixel intensities in each block to compute a binary code. Such operators are for example the LBP and the MCT encoding described in Table 2.1.

The **multi-block feature pool** consists of all the multi-block patterns that can be generated to fit a given model size. This pool is constructed by varying independently the top-left coordinates and the cell sizes and applying different encoding operators.

Let $\mathfrak{D}(W, H)$ be the exhaustive set of multi-block features using models of the size $W \times H$ pixels. Then the following constraints must be satisfied: $0 \leq l < W - c_x n_x$ and $0 \leq t < H - c_y n_y$, for a particular feature parametrized by $(t, l, c_x, c_y, n_x, n_y)$. The number of valid features increases at a cubic rate with the model size, because the multi-block features are collected from any location and scale. Thus the size of the feature pool increases rapidly as we increase the size of the model, a summary of this is provided in Table 4.1. It can also be noticed that the number of features per pixel increases with the model resolution which indicates that the feature set consists of many redundancies.

Boosting a model $\mathbf{f}$ results in a relatively small number of features selected to form weak learners, while many others are discarded. We shall denote with $\mathfrak{D}(\mathbf{f})$ the selected (boosted) multi-block

features for the model $\mathbf{f}$.

For example, boosting a high resolution 80×96 pixel model requires processing approximatively 6.1 million EMB-LBP features at each round. This is unmanageable, even if the boosting algorithm is itself scalable. However, boosting 20×24 models is fairly fast. This is because approximatively 20,000 features can be easily pre-computed and stored in memory for a large number of samples (e.g. of the order of 100,000). Then, the boosting algorithm is performing simple indexing in the data structures where the features are stored.

Coarse-to-fine multi-block feature selection

The proposed **coarse-to-fine multi-block feature selection** algorithm is based on a key assumption. We assume that the boosted multi-block features are *close* in location and scale, when upscaled, to the optimal features selected for the model of twice the resolution. This is equivalent to assuming that *close* multi-block features have similar performance. Intuitively, the set of all multi-block patterns consists of many overlapping and redundant features and thus projecting them to a lower resolution may not significantly degrade the performance of the boosted model.

We propose to identify the location and the scale of the multi-block features at the coarse resolution and then to iteratively refine the size and the ellongation of the selected features at higher resolutions. There are two distinguishable steps.

1. The **initialization** consists of exhaustively boosting features from a coarse enough scale such that boosting is feasible. The selected features provide a rough approximation of the location and scale of the optimal boosted features at the highest resolution.

2. The **refinement** iterations boost a relatively small collection of features at twice the resolution of the previous step. This collection of features is constructed using the selected features at the previous step, projected to twice the resolution, and varying independently their size with a fixed increment. This increment is halved at each refinement iteration to obtain better approximations of the optimal features.

More precisely, let $MB_p = (t, l, c_x, c_y, n_x, n_y) \in \mathfrak{D}(\mathbf{f})$ be a set of boosted multi-block features at the resolution of $W_p \times H_p$ pixels for the projection p. This feature set is then projected to $MB_{p+1} = (2t, 2l, 2c_x, 2c_y, n_x, n_y)$ using the higher resolution model of size $2W_p \times 2H_p$ pixels. We construct 9

additionally higher resolution patterns centered on the set of patterns from MB_{p+1}, with cell sizes of $\{2c_x - 1, 2c_x, 2c_x + 1\} \times \{2c_y - 1, 2c_y, 2c_y + 1\}$. This set of patterns, illustrated in Fig. 4.1 (b-j), is denoted as $\mathbf{MB}_{p+1}$. We thus construct at the resolution $2W_p \times 2H_p$ up to 9 times more patterns than the boosted ones at the resolution $W_p \times H_p$. This procedure is referred to as **feature projection**. Next the boosted projected features shall be considered as better approximations of the optimal multi-block features. The state diagram is presented in Fig. 4.2.

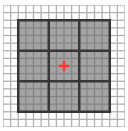
We formalize the proposed coarse-to-fine feature selection method in Algorithm 6. The current set of features is denoted with $\mathfrak{D}$. At first, $\mathfrak{D}$ is initialized with the exhaustive set of multi-block features for the coarsest resolution (step 2). Then, $\mathfrak{D}$ is iteratively updated by projecting (at twice the resolution) the selected features at the previous resolution (steps 5-6). Finally, a model of high resolution $W_0 2^P \times H_0 2^P$ is produced after P projection steps.

Algorithm 6 Coarse-to-fine multi-block feature selection.

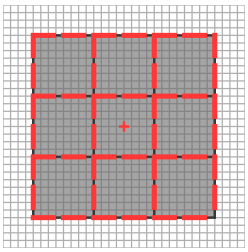
1: Given: coarsest resolution $W_0 \times H_0$, projections P
2: Exhaustive features: $\mathfrak{D} \leftarrow \mathfrak{D}(W_0, H_0)$
3: Boost model: $\mathbf{f}_0 \leftarrow boost(\mathfrak{D})$
4: **for** $p = 1$ **to** $p \leq P$ **do**
5: Select features: $\mathfrak{D} \leftarrow \mathfrak{D}(\mathbf{f}_{p-1})$
6: Project features: $\mathfrak{D} \leftarrow project(\mathfrak{D})$
7: Boost model: $\mathbf{f}_p \leftarrow boost(\mathfrak{D})$
8: **end for**
9: **return** $\mathbf{f}_P$ of size $W_0 2^P \times H_0 2^P$ pixels

The algorithm is expected to produce models of similar performance to the model trained with the exhaustive set of features at the highest resolution $W_0 2^P \times H_0 2^P$. However, the coarse-to-fine feature selection is significantly faster. For example, let us consider the case of training a 80×96 model using 1024 weak learners and EMB-LBP features. The exhaustive pool of features consists of $|\mathfrak{D}(80, 96)| \approx 6,100,000$ features (see Table 4.1). However, if we use $P = 2$ projection iterations, the model is first trained with $|\mathfrak{D}(W_0 = 20, H_0 = 24)| \approx 19,100$ features. Then, the selected 1024 features are projected twice to obtain roughly 9,000 features at resolution 40×48 and 80×96 pixels respectively. Thus, the coarse-to-fine feature selection method process $37,100$ features in total, which is approximatively 160 times faster than the exhaustive approach.

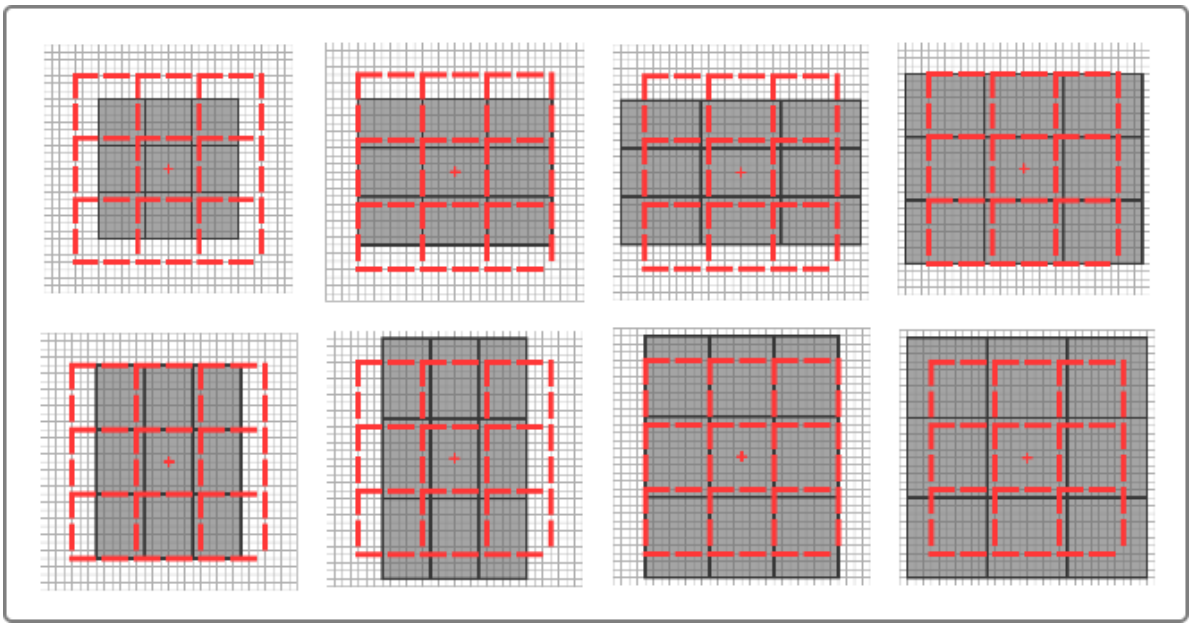
It is important to notice that the proposed feature selection algorithm is generic in nature. Any multi-block features (e.g. MB-LBP, EMB-LBP, MCT) that respect the definition above can be successfully processed.



(a) original

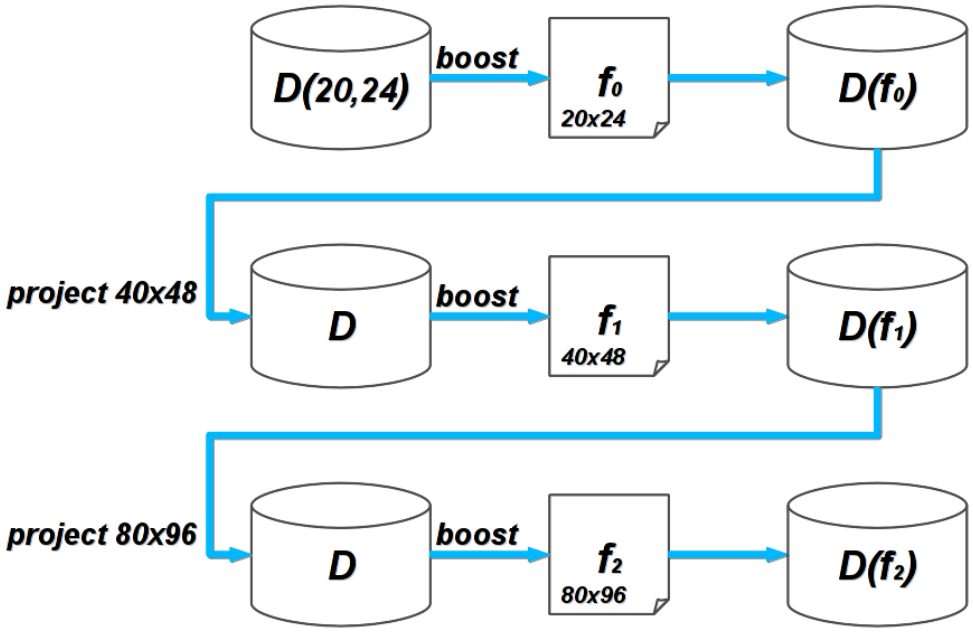


(b) projected pattern



(c) additional projected patterns

Figure 4.1. Illustration of the proposed coarse-to-fine feature projection. (a) The coarse original multi-block pattern (MB_c). (b) The projected multi-block pattern to twice the resolution (MB_h). (c) The set of additional patterns, constructed by varying the cell size independently for each axis (**MB**$_h$). The center of the patterns is displayed with a red cross, while the upscaled pattern is contoured with a dashed red line.



(a)

Figure 4.2. Illustration of the proposed coarse-to-fine feature selection algorithm. The algorithm maintains a set of features that is initialized with the exhaustive set at the coarsest model resolution. Then iteratively, the feature set is pruned by boosting and projected to twice the model resolution.

4.2 Sampling and bootstrapping training data

Another aspect of speeding-up the boosting algorithm is to sample a representative small dataset when a large pool of training samples is available. The training pool consists of sub-windows (patches) of fixed size in pixels (the model size), collected at different locations and scales of the input images to achieve robustness to simple transformations of the input. In practice, this is performed by scaling each input image to form a pyramid of images. Then, at each scale the samples are collected by shifting the samples (translation) using a fixed step. This scanning process generates a large number of samples from a relatively small number of images (e.g. billions of samples from a hundred 640×480 images) if performed densely in location and scale.

It is not feasible to boost models using such large collections of training samples, even if the number of features is small. The solution is to train a model using a relatively small and representative subset of all potential training samples. This section describes such a practical solution.

Let $\mathbf{Z}$ be a potentially very large set of training samples. Each sample has a type out of K total types. For example, in the case of face detection there are two such types: the face samples and the

background. The sampling pool can thus be decomposed into distjoint sets: $\mathbf{Z} = \bigcup_k \mathbf{Z}_k$, of samples having the same type $k \in \{1, .., K\}$. We use the notation $\mathbf{Z}_z \subseteq \mathbf{Z}$ to denote the set of samples having the same type as the sample $z \in \mathbf{Z}$.

Let $\epsilon(z|\mathbf{f})$ be an error function that measures the accuracy of the prediction matching the target for the sample z using the model $\mathbf{f}$. Usually, this is non-continuous function that it is hard to optimize. The optimization criteria is chosen instead to be the smooth and differentiable loss function l. We use the error function to compute the probability of bootstrapping a given sample. The key idea is to favor samples with large errors, thus forcing the model to learn these samples in the next iteration. By iteratively adding the mis-predicted samples to the training data, the large training pool is systematically explored and the model is trained with representative samples.

The goal of the sampling procedure is to select a small subset of samples, that respect the following two concepts.

1. The subset is **balanced** over the intrinsic type. This is especially useful when some types of samples appear significantly more frequent than the others (Viola and Jones, 2002). For example, in the case of face detection the number of background samples are of several order of magnitude more frequent than the face samples. Training a model using uniformly distributed data biases the model towards the most frequent type of samples, because their contribution to the loss (both expectation and variational) is greater.

2. The subset is **representative** to ensure that sampling does not degrade the performance. This is performed by iteratively **bootstrapping** the pool of samples. Bootstrapping is widely used for building state-of-the-art cascade of face classifiers (Viola and Jones, 2001; Froba and Ernst, 2004; Zhang et al., 2007). The training samples are augmented or replaced at each step (stage of the cascade) with the ones mis-classified at the previous step. We generalize these ideas to non-cascaded strong learners and to regression problems. Similarly, we augment at each bootstrapping step the training samples with the ones having large errors, with high probability. The number of boosting rounds is doubled at each step, because the training data becomes more challenging.

We distinguish between uniform and error-based sampling methods. The **uniform sampling** is a balanced method such that each type is approximatively equally distributed in the resulting

data. Thus the probability of selecting a sample z is thus defined to be proportional to the number of samples having that type:

$$p_u(z \in \mathbf{Z}|N) = \frac{N}{K}\frac{1}{|\mathbf{Z}_z|}, \tag{4.1}$$

where N is the number of samples to select. If the probability function is greater than unit, than that sample can be selected multiple times. The set of selected samples is denoted as: $\{\sim p_u(\cdot\,|N)\}$.

The **error-based sampling** takes into consideration the error obtained with the current model $\mathbf{f}$, such that mis-predicted samples have thus higher chances of being selected:

$$p_e(z \in \mathbf{Z}|N, \mathbf{f}) = \frac{N}{K}\frac{\epsilon(z|\mathbf{f})}{\sum\limits_{s \in \mathbf{Z}_z} \epsilon(s|\mathbf{f})}. \tag{4.2}$$

The set of selected samples is denoted as: $\{\sim p_e(\cdot\,|N, \mathbf{f})\}$.

The proposed bootstrapping algorithm is presented in Algorithm 7. Let N_b^t, N_b^v and R_b be the number of the training samples, the validation samples and the boosting rounds respectively, at the bootstrapping step $b \leq B$. Z_b^t and Z_b^v are the training and the validation samples respectively. Clearly $N_b^t = |Z_b^t|$ and $N_b^v = |Z_b^v|$ respectively. We distinguish the particular case of $B = 0$, when no bootstrapping steps are performed, as *one-shot* boosting.

There are two basic bootstrapping operations: sampling and training. It can be noticed that the validation dataset is re-sampled at each bootstrapping step to reduce the biasing effect. The number of validation samples is preserved at each step: $N_b^v = |Z_b^v| = |Z_0^v| = N_0^v$, while the number of training samples increases linearly: $N_b^t = |Z_b^t| = (b+1)|Z_0^t| = (b+1)N_0^t$.

Algorithm 7 Bootstrapping training samples.

Given: $N_0^t > 0,\ N_0^v > 0,\ R_0 > 0,\ B \geq 0$
Uniform sample training data: $Z_0^t \leftarrow \{\sim p_u(\cdot\,|N_0^t)\}$
Uniform sample validation data: $Z_0^v \leftarrow \{\sim p_u(\cdot\,|N_0^v)\}$
Boost model: $\mathbf{f} \leftarrow boost(Z_0^t, Z_0^v, R_0)$
for $b = 1$ **to** $b \leq B$ **do**
    $R_b = 2R_{b-1}$
    $N_b^t = N_{b-1}^t + N_0^t$
    Bootstrap training data: $Z_b^t \leftarrow Z_{b-1}^t \bigcup \{\sim p_e(\cdot\,|N_b^t, \mathbf{f})\}$
    Uniform sample validation data: $Z_b^v \leftarrow \{\sim p_u(\cdot\,|N_0^v)\}$
    Boost model: $\mathbf{f} \leftarrow boost(Z_b^t, Z_b^v, R_b)$
end for
return $\mathbf{f}$ of up to $R_0 2^B$ weak learners

The number of boosting rounds increases exponentially, partially motivated by the design of the cascades used for real-time face detection (Viola and Jones, 2001). The first stage of the cascades usually consists of few weak learners, while the next stages rapidly reach hundreds or even thousands of weak learners. However, we retrain at each bootstrapping step the strong learner with the augmented training samples (which includes the selected samples at the previous step). This way the model does not forget the previous samples, but trains with a better distribution of the training samples over the input signal space.

4.3 Summary

This chapter introduced a generic method for boosting high resolution models using large pools of samples. First, we have introduced the coarse-to-fine multi-block feature selection algorithm. The algorithm iteratively projects the currently boosted features to higher resolutions, instead of processing the exhaustive feature set. Thus, the computation is concentrated on refining a small fraction of all possible features that are selected by boosting at coarser levels. Second, we detailed the proposed sampling and bootstrapping method to build efficiently a representative training dataset from a very large pool of samples. The next chapter describes the experimental results for the face detection task.

Chapter 5

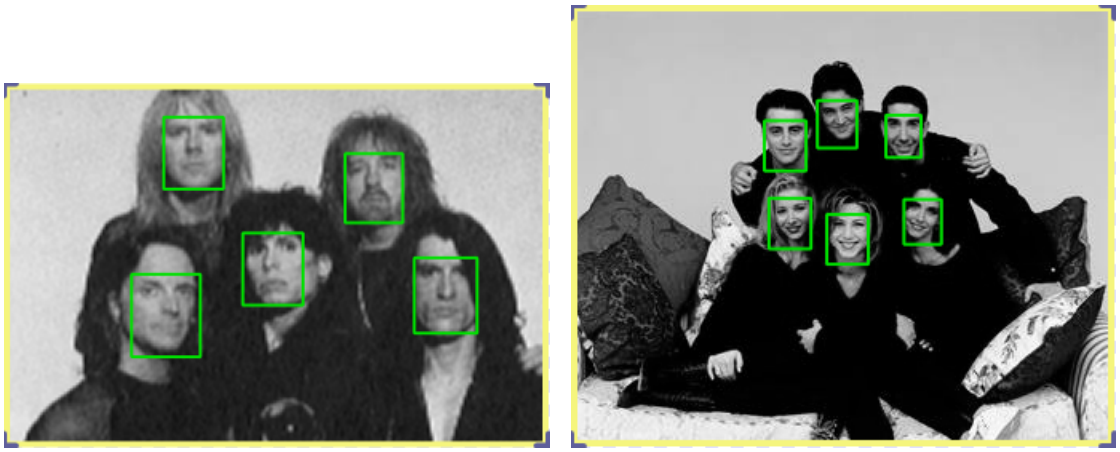
Application to face detection

In this chapter we apply our boosting framework to the task of face detection. Face detection is a binary classification task that consists of finding the position of all the faces, if any, in an image. Two components are usually required: a classifier and a search algorithm. The search (or scanning) algorithm forms sub-windows (or samples) at different locations and scales which are fed to the classifier. The sub-windows labeled as positive samples are considered as final detections. Usually a clustering algorithm (e.g. non-maxima suppression, averaging the overlapping regions, mean shift) is run on these detections to reduce the number of multiple detections.

The face detections can then be further processed to accurately locate the facial features of interest (e.g. eye centers, mouth corners, nose tip). The facial feature locations are then typically used to geometrically normalize detections to improve face recognition or to initialize high level systems (e.g. facial expression analysis). This means that efficiency is an important aspect of any face detection system.

5.1 Background

Recently there has been a great interest in real-time face detection systems. Their speed depends mostly on the speed of the classifier to evaluate a sub-window. These systems are usually built using boosted classifiers (Viola and Jones, 2001; Zhang et al., 2007) because of their potential computational efficiency while providing state-of-the-art performance. Another important factor is the



(a) (b)

Figure 5.1. Illustration of two typical images to evaluate face detection models. The ground truth face locations are overimposed with green rectangles.

speed to compute the features. The fastest features are evaluated in constant complexity at any location and scale, for example: Haar-like features (Viola and Jones, 2001) and MCT (Froba and Ernst, 2004) or multi-block LBP codes (Zhang et al., 2007).

Typical images to process are illustrated in Fig. 5.1. The objective of a face detection systems is to produce the correct number of face detections and each detection to match as close as possible the associated ground truth location. The boosted model has a single output, which is ideally positive for faces and negative for background samples. The detections are validated using the Jaccard similarity index (Jaccard, 1901):

$$J(D,G) = \frac{D \cap G}{D \cup G}, \tag{5.1}$$

which is proportional to the amount of the overlap between the detection D and the ground truth G. If the overlap, expressed as a percentage, is greater than 50%, then D is considered as a detection, otherwise, it is considered a false alarm (mistake).

Face detection models are evaluated using the **free-response receiver operator curve** (FROC). This is computed by changing the detection threshold of the model. Each threshold value produces two values that are plotted as a point on the curve. The first is the detection rate (DR). It measures the percentage of the faces correctly detected, which is the number of ground truth face locations that overlap more than 50% with at least a detection. The second is the number of false

alarms (FA) [1], which is the number of detections that does not overlap more than 50% with any ground truth face location. The higher the resulting FROC, the better the model: it detects more faces at the same number of false alarms.

5.2 Experimental protocol

The face detection task is addressed using a low-resolution model of 20×24 pixels. This is to cope with small faces from benchmark face datasets and because it is of sufficient size to achieve good performance. The model is vertically ellongated to cover facial features situated in the lower part of the face (e.g. mouth corners).

5.2.1 Training and validation protocol

The training and validation face datasets consist of well-known collections of images. The face detector was trained using approximately 10,000 face images from the XM2VTS (Messer et al., 1999), BANCA (Bailly-Bailliére et al., 2003) and CMU-PIE (Sim and Baker, 2003) datasets for the positive samples. The background, or negative, samples consist of 1,500 images from the CALTECH-101 (Fergus and Perona, 2007) and the CALTECH (Weber) background datasets. Some typical training face images are shown in Fig. 5.2.

We use the XM2VTS, CMU-PIE and CALTECH-101 as training datasets and BANCA and CALTECH as validation datasets. From these images we build two large pools of 200 million and of 60 million training and validation samples respectively (a large part being background) using a fine discretization of location and scale of two pixels. The training and the validation samples were sampled from this pool. The positive samples consists of sub-windows covering at least 80% of the ground truth face location, while the negative samples were collected only from images without no faces.

The model was trained using $R = 1024$ boosting rounds and 7 bootstrapping steps ($B = 7$). We sample 80,000 training samples and 80,000 validation samples ($N_0^t = 10,000, N_0^v = 80,000$). We have chosen the logistic loss $l(y, f) = log(1 + exp(-yf))$ to optimize with the expectation loss formulation. Obviously, the model has one output ($O = 1$). The associated error function is

[1] In contrast, the receiver operator curve (ROC) measures the DR versus the false alarm rate (FAR) and it is sometimes erroneously used to denote FROC face detection results (Viola and Jones, 2001).

(a) XM2VTS (b) XM2VTS

(c) BANCA (d) BANCA

(e) CMU-PIE (f) CMU-PIE

Figure 5.2. Illustration of typical training face images from the XM2VTS (a, b), BANCA (c, d) and CMU-PIE (e, f) datasets. The ground truth face locations are overimposed with green rectangles.

Setup	
Train + validation	80,000 + 80,000 samples
Resolution	20×24 ($P = 0$)
Outputs	$O = 1$
Features	EMB-LBP
Rounds	$R = 1024$
Loss function	$l(y, f) = log(1 + exp(-yf))$
Error function	$\epsilon(\mathbf{z}\|f) = \epsilon(\{\mathbf{x}, y\}\|f) = \mathbb{1}(yf(\mathbf{x}) \leq 0)$

Model name	**Description**
BOOT	$B = 7$ bootstrapping steps
SHOT	$B = 0$ bootstrapping steps
EPT	Expectation loss formulation
VAR	Variational loss formulation $\lambda \in \{0.0, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0\}$

Table 5.1. The upper half defines the common parameters for all models, while the lower half defines the parameters used for training the face models that we evaluate.

$\epsilon(z|f) = \epsilon(\{x, y\}|f) = \mathbb{1}(yf(x) \leq 0)$. The error function is used to bootstrap the mis-predicted samples and to tune on the validation dataset the number of rounds and the regularization factor λ of the variational loss formulation. The selected training samples are sampled to be balanced over the two types ($K = 2$): positive - face and negative - background.

Multiple detections are integrated using non-maxima suppression. This is performed iteratively. At each iteration, the detection with the highest score is kept, while all the detections that overlap with it are removed. Finally, the detections are thresholded.

For these experiments we have trained several models to assess the impact of the proposed bootstrapping method (**BOOT** vs. **SHOT**) and of the variational loss formulation (**VAR** vs. **EPT**). The parameters of all the evaluated models are presented in Table 5.1.

5.2.2 Testing protocol

We present face detection results on two test datasets. The first is the MIT+CMU dataset (Rowley et al., 1998) that contains 130 images, with multiple, sometimes very small, degraded faces or without any faces, taken in different environments (indoor and outdoor). This is the most used testing dataset for face detection and it is considered challenging. We have compared our model with the state-of-the-art reported results of FCBoost (Saberian, 2010) and Viola and Jones (Viola and Jones, 2001).

The second dataset - BioID (Jesorsky et al., 2001), contains 1520 images with a single face per image. This dataset is considered less challenging and it is mainly used as benchmark for facial feature localization. We have compared our model with the reported resuls of Froba and Ernst (Froba and Ernst, 2004).

5.3 Results and discussions

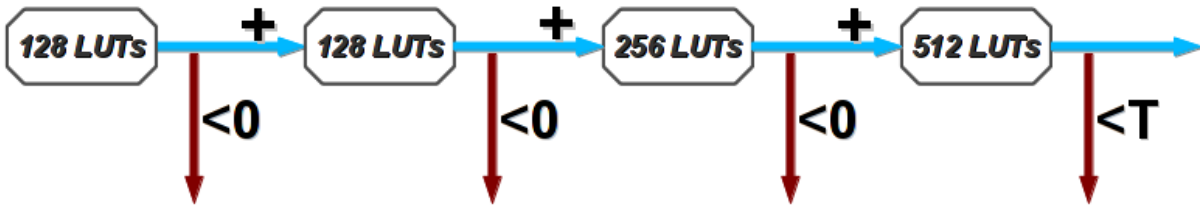
In this section we examine several aspects of our proposed framework. First, we evaluate the benefit of splitting the model into levels and we evaluate the effectiveness of performing bootstrapping, of sharing features and of the variational loss formulation. Second, we analyze the selected EMB-LBP features by our framework.

5.3.1 Performance analysis

We illustrate in Fig. 5.7 some typical face detection results for the MIT+CMU dataset.

Evaluating levels

The first set of experiments consist of evaluating the speed of the face detector. For this, we split the boosted model into **levels**, similar to the stages of the boosted cascades (Viola and Jones, 2001). Each level consists of twice the number of look-up-tables as the previous one. After each level, the current sample score is thresholded with zero. If it is below then the sample is rejected as being a false alarm and otherwise it is processed by the next levels (see Fig. 5.3). The default threshold of zero could be further tuned similar to the cascade process described in (Viola and Jones, 2001)



(a)

Figure 5.3. Illustration of splitting of a 1024 look-up-tables model using 3 levels. The sub-window is rejected as false alarms if the current cumulated level score is negative. If all the levels are passed, the final score is finally thresholded with an optimized value T. The sub-window rejection paths are represented with red, while the sub-window acceptance paths are represented with blue.

DataSet	Levels 4	Levels 6	Levels 8	Levels 10	FCBoost	Viola&Jones
MIT+CMU	69.4	21.5	8.9	**7.12**	7.2	8
BioID	69.3	21.5	8.4	**6.5**	-	-

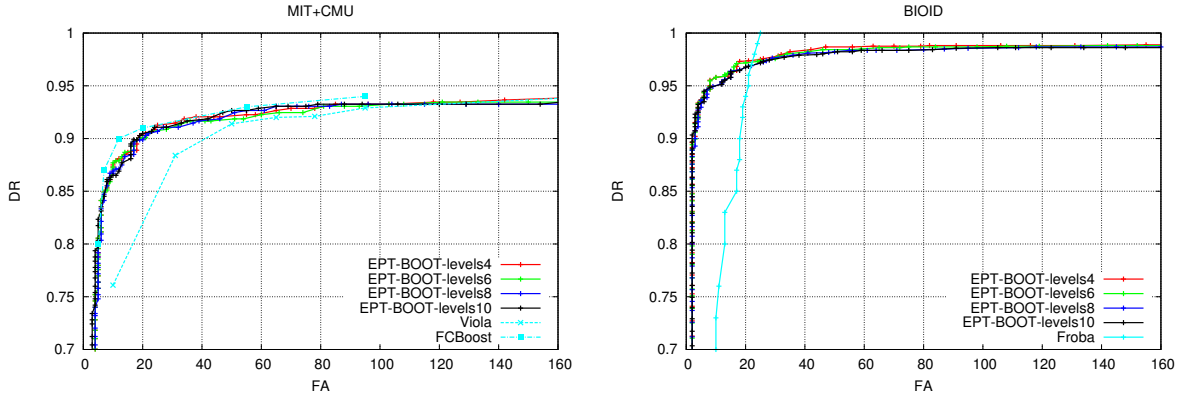
Table 5.2. The number of look-up-table evaluations (and multi-block feature computations) per sub-window for different number of levels using the **EPT-BOOT** model.

to achieve the maximum true rejection rate of the background samples while accepting an imposed minimum rate of true acceptance. However, this is beyond the scope of this work and we simply present results for the ad-hoc splitting into levels.

The number of levels, thus the number of splits of the strong learner, is directly proportional to the speed of the detector. This is because a background sub-window (that account for the vast majority of test samples) has a higher chance of being rejected after significantly fewer look-up-table evaluations.

We have plotted the FROC curves obtained for various number of levels in Fig. 5.4. The average number of LUT evaluations, and implicitly of multi-block features computed, per sub-window is displayed in Table 5.2. It can be noticed that the performance is not very sensitive to the number of levels, in the worse case the DR decreases by 2% with the same number of false alarms. The proposed ad-hoc split of the strong learner achieves similar speed and performance as FCBoost (Saberian, 2010), which is one of the fastest state-of-the-art face detectors. The authors report for FCBoost 7.2 weak learner evaluations on average per sub-window, similar to the results we present for 10 levels. The proposed boosted model is thus performing face detection in **real-time**, without sacrificing performance at comparible performance with state-of-the-art systems.

We consider there are two key aspects that motivate empirically the proposed splitting of the



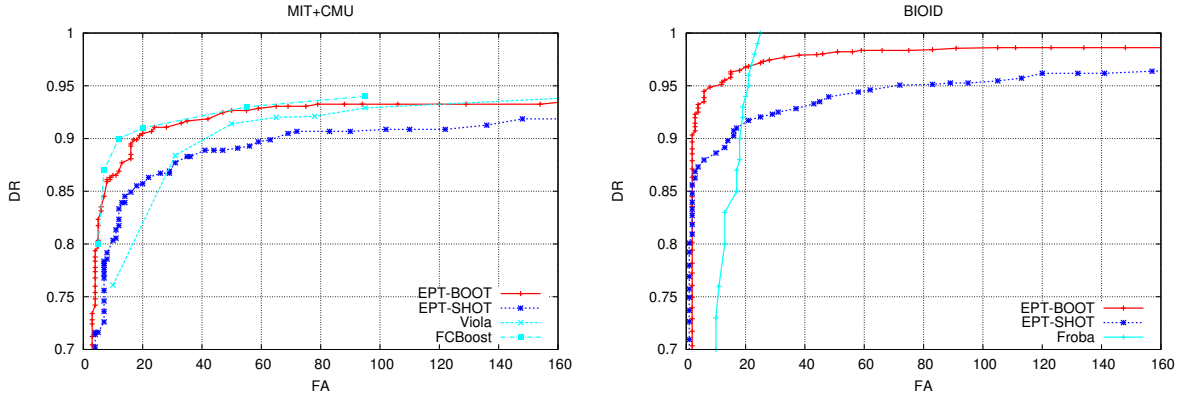
(a) MIT+CMU (b) BIOID

Figure 5.4. The face detection FROC curves for the MIT+CMU (a) and the BIOID dataset (b) using the **EPT-BOOT** model and different number of levels.

strong learner in levels. The first is that the loss decreases faster in the first boosting rounds and significantly slower afterwards. Thus, any starting sequence of weak learners is a good approximation of the overall model. Taking a decision earlier (in the number of weak learners) is thus not degrading the performance too much. The second regards the sliding-window face (object) detection approach. This method evaluates multiple overlapping sub-windows and thus builds a redundant set of potential true detection for each face. The face is not detected only if all overlapping sub-windows are rejected by the classifier. This is unlikely to happen because there is an in-built redundancy due to the fact that the scanning process will have many sub-windows which overlap with the same face. The detector is thus robust to false rejection mistakes made earlier in the levels, while the false alarms are efficiently pruned using non-maxima suppression.

Evaluating bootstrapping

Next, we examine the potential benefit of bootstrapping the training samples. The training data consists of 80,000 samples selected uniformly and then refined using the error function out of a pool of 200 million samples. The **EPT-SHOT** model samples the same amount of training data, but uniformly at once without investigating the performance of the model accross the sample pool. As illustrated in Fig. 5.5, the proposed bootstrapping method greatly improves the performance of the detector by 5% DR on average for the same fixed number of false alarms.



(a) MIT+CMU (b) BIOID

Figure 5.5. The face detection FROC curves for the MIT+CMU (a) and the BIOID dataset (b) using the **EPT-BOOT** and the **EPT-SHOT** models with 10 levels.

Evaluating the loss formulation

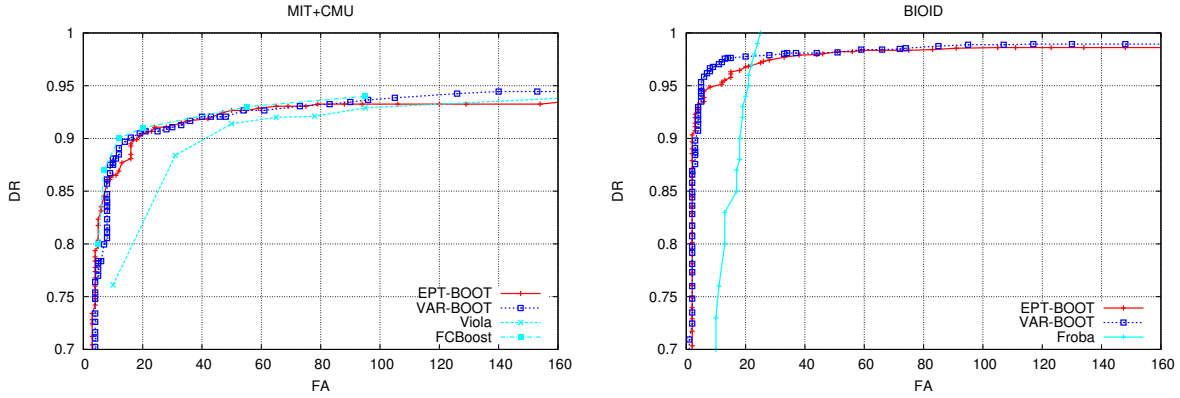
The final set of experiments consists of evaluating the variational loss formulation compared to the widely used expectation formulation. The variational model **VAR-BOOT** was trained similarly to the **EPT-BOOT** model, with the difference that the regularization factor λ was tuned on the validation dataset. We have tried the following values: $\lambda \in \{0.0, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0\}$ independently for each bootstrapping step.

The two models are compared in Fig. 5.6. There is a slight improvement of 1-2% DR at same number of false alarms for the MIT+CMU dataset. The difference is higher, up to 3% DR, for the BioID dataset in the region of few false alarms. However, the rather modest increase in performance must take into account the training time that is increased by a factor of 8.

The optimal regularization factors at each bootstrapping step are presented in Table 5.3. The model selects the value of 2 as the optimal parameter for most bootstrapping steps.

5.3.2 Feature selection analysis

Boosting is often used as feature selection, because the features associated with the selected weak learners can be interpreted as the most useful features for some specific task. We have analyzed the selected multi-block patterns to assess: the size (the (c_x, c_y) parameters) and the encoding type (LBP, tLBP, dLBP, mLBP) of the selected features.



(a) MIT+CMU (b) BIOID

Figure 5.6. The face detection FROC curves for the MIT+CMU (a) and the BIOID dataset (b) using the **EPT-BOOT** and the **VAR-BOOT** models with 10 levels.

Boosting rounds	**Optimal regularization factor**
8	$\lambda_0^* = 1.0$
16	$\lambda_1^* = 5.0$
32	$\lambda_2^* = 2.0$
64	$\lambda_3^* = 2.0$
128	$\lambda_4^* = 2.0$
256	$\lambda_5^* = 2.0$
512	$\lambda_6^* = 1.0$
1024	$\lambda_7^* = 2.0$

Table 5.3. The number of boosting rounds and the optimal regularization factor λ for each bootstrapping step when training the **VAR-BOOT** model.

The **EPT-BOOT** model consists of multi-block patterns of 42 different cell sizes ($(c_x, c_y) \in \{1,2,3,4,5,6\} \times \{1,2,3,4,5,6,7\}$). However, the distribution of the cell sizes is very skewed, because most of the them account for less than 3% of weak learners. The smaller patterns are selected more frequently, which is not surprising considering the low resolution of the face model. The selected features are most frequently of size: 3×3 (23%), 6×3 (9.8%), 3×6 (7%), 6×6 (5.4%), 6×9 (4.7%), 9×3 (4.5%), 9×6 (3.5%), 9×9 (3.2%) and 3×9 (3%).

The analysis of the LBP encodings have shown that the dLBP feature is less informative than all the other three as it is not selected at all. This matches the experimental findings previously reported for face detection using EMB-LBP features (Trefny and Matas, 2010). However, the ratio of the three selected encodings is significantly different. The LBP and mLBP encodings account for just 4.5% and 18.7% respectively, while the tLBP encoding is used by 76.8% of the weak learners. This contrasts with (Trefny and Matas, 2010), where the authors reported more balanced ratios of 30-40% approximately.

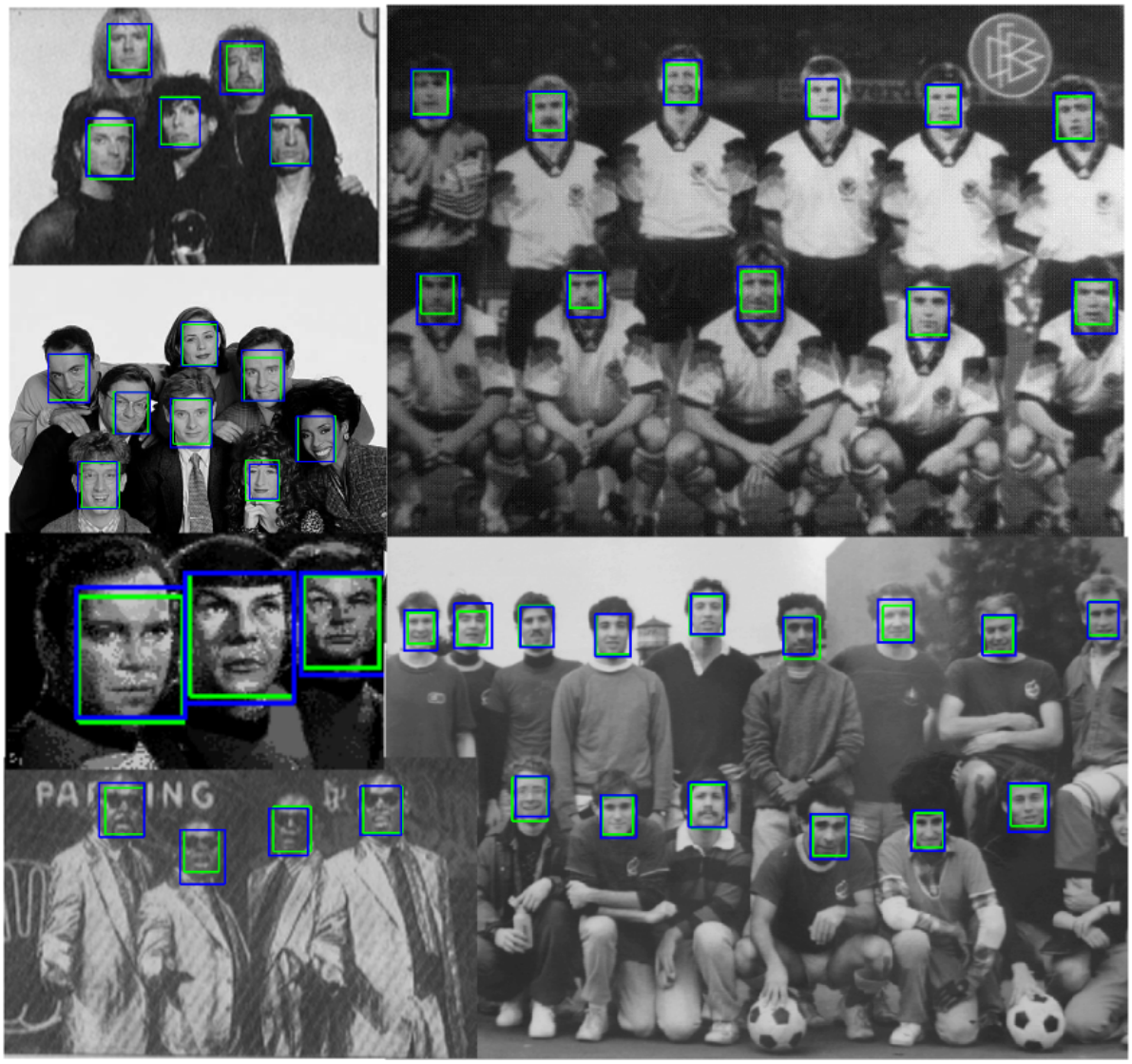
5.4 Summary and concluding remarks

This chapter studied the proposed boosting framework for the task of frontal face detection. We have studied the performance impact of several aspects of the proposed model: the ad-hoc splitting of the model into levels to speed-up evaluation, the bootstrapping and the variational loss formulation. Overall, we have obtained a real-time face detector with similar performance to the state-of-the-art systems proposed in literature.

The experimental findings can be summarized as:

1. Splitting the boosted classifier into levels speeds-up significantly the evaluation, without deteriorating the performance. The fastest detector processes on average 7.12 look-up-tables for each sub-window on the MIT+CMU dataset, at state-of-the-art performance. This is close to the highest speeds reported in literature (Saberian, 2010; Viola and Jones, 2001). However, the thresholds of each level were not tuned and set to zero by default which may not be the optimal compromise between performance and speed. It is surprising though that such a simple splitting heuristic works as well as boosted cascades (Saberian, 2010).

2. The proposed bootstrapping method improves the detection rate by 5% on average at the



(a)

Figure 5.7. Illustration of some face detection results on the MIT+CMU dataset. The ground truth face locations are represented with green and the detections with blue.

same number of false alarms, for both test datasets, MIT+CMU and BioID, compared with the one-shot model. The bootstrapped model builds iteratively a training dataset that uniformly samples positive (face) and negative (background) samples and is representative for the large sampling pool. In contrast, the one-shot model uniformly samples the training data once.

3. The variational loss formulation produces a modest increase in performance compared to the expectation loss formulation. This may be because the training dataset is challenging and boosting does not overfit, thus a regularized model presents no significant advantage. However, training a variational model is approximately 8 times more time consuming than an expectation model, because the regularization factor λ needs to be tuned on the validation dataset.

The next chapter describes the experimental results for the facial feature localization task.

Chapter 6

Application to facial feature localization

In this chapter we apply our boosting framework to the task of facial feature localization. Face detections are processed to accurately locate the facial features of interest (e.g. eye centers, mouth corners, nose tip). Facial feature locations are typically used to geometrically normalize detections to improve face recognition or to initialize high level systems (e.g. facial expression analysis). We will show that this task can be cast as a multivariate regression task.

6.1 Background

Facial feature localization methods can be broadly classified into global and local methods. The global methods process the face region as a whole and in a single step. For example, in Everingham and Zisserman (2006) the authors evaluate basic regression, Bayesian and classification approaches. While in Cristinacce and Cootes (2003) the facial features are located using individual AdaBoost classifiers and a shape constrain method to eliminate ambiguities. The local methods iteratively refine the current location estimate using local appearance measurements and global shape constraints. Active Shape Models (ASM) and Active Appearance Models (AAM) have been widely used for localizing facial features or anatomical key features in medical images Cristinacce and Cootes (2006, 2007, 2008). Other interesting recent work includes the boosted regression and

(a) (b)

Figure 6.1. Illustration of two typical images to evaluate facial feature localization models. The ground truth face locations are overimposed with green rectangles, while the ground truth facial features are displayed with blue crosses.

graph models proposed in Valstar and Binefa (2010) and the cascaded pose regression model from Doll and Pietro (2010).

The goal of facial feature localization models is to predict the location of the facial features as close as possible to the ground truth locations. Fig. 6.1 presents typical images used to evaluate facial feature localization models. Different sets of facial features are considered depending on the end application. For example, accurately predicting the eye locations may be sufficient for geometrically normalizing faces for recognition, but expression analysis applications may require a larger set of facial features.

The performance of a facial feature localization model is measured using the **average point-to-point distance** (Cristinacce and Cootes, 2003) between predictions and ground truth locations, normalized to the distance between the eye centers. More precisely, let F be a set of facial features having the G_i ground truth locations. Let D_i be the model predictions for each point. The boosted model has $2 \times F$ outputs, that concatenate the vertifical and the horizontal coordinates of each facial feature. If the distance between the eye centers is $\Delta = d(G_{leye}, G_{reye})$, then the localization error of the model is:

$$E(D, G) = \frac{1}{F\,\Delta} \sum_{1 \leq i \leq F} d(D_i, G_i), \tag{6.1}$$

where d is the Euclidean distance between two-dimensional points. Clearly, the more accurate a model is, the smaller the measure error E. We shall present results by restricting E to the interval

$[0, 0.30]$. Values higher than 0.30 usually correspond to random or unusable predictions.

6.2 Experimental protocol

This section describes the training and the testing protocols. We also detail the parameters used to boost these localization models.

6.2.1 Training and validation protocol

The large publicly available **CMU MultiPIE** dataset (Gross et al., 2010) is used to train the localization models. We annotated [1] the CMU MultiPIE dataset with 16 facial features (eye centers, eye corners, nose tip, mouth corners, bottom and top lip, chin and eye brows corners) as illustrated in Fig. 6.2 (f).

The CMU MultiPIE dataset consists of more than 750,000 images of 337 people recorded in office environments, with close to uniform background. The recordings were performed in four sessions. Multiple images for each subject (client) were collected using 19 illumination conditions, 15 view points and various facial expressions. We use five frontal (and close to fontal) poses to train and to evaluate frontal face models, illustrated in Fig. 6.2. We sample randomly five images for each client, pose, session and facial expression. Next, we split the images into training, validation and test datasets as specified in Table 6.1. The protocol is chosen as to have distinct clients in each dataset, this is to ensure that the trained model generalizes well.

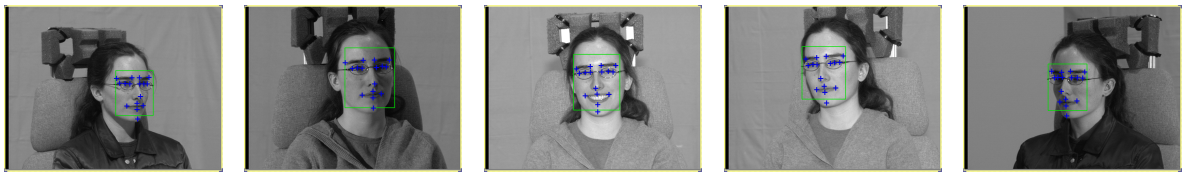
We boost high resolution models, of size 80×96, to predict the location of various sets of these points. The training dataset for each set of facial features consists of sub-windows that overlap at least 60% with the ground truth face location so that the localization model is robust to large variations in location and scale of the face detections relative to the ground truth. The pool of training samples is built using a fine discretization of location and scale, similar to the one used to generate samples for the face detection case.

The model is trained using 1024 boosting rounds ($R = 1024$) and 3 bootstrapping steps ($B = 3$). We sample 80,000 training samples and 80,000 validation samples ($N_0^t = 20,000, N_0^v = 80,000$). The multi-block features were projected twice using the proposed coarse-to-fine feature selection,

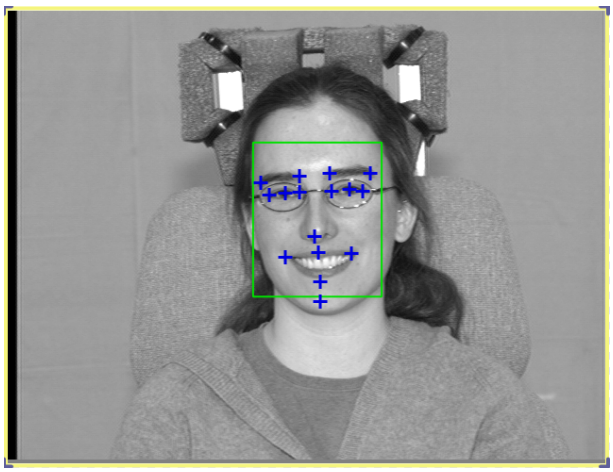
[1] The CMU MultiPIE annotations will be available shortly at `http://www.idiap.ch/resource/biometric/`

Dataset	Client IDs
Train	1, 7, 12, 13, 16, 21, 24, 25, 26, 30, 31, 32, 37, 39, 45, 51, 58, 59, 60, 61, 63, 65, 66, 72, 73, 75, 77, 81, 82, 84, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 98, 99, 101, 109, 113, 114, 119, 120, 121, 130, 134, 135, 136, 140, 141, 142, 144, 146, 147, 148, 151, 152, 153, 154, 155, 158, 159, 160, 162, 163, 164, 165, 166, 171, 172, 173, 174, 176, 179, 180, 182, 183, 187, 189, 195, 197, 200, 201, 204, 206, 207, 210, 211, 212, 214, 215, 216, 217, 218, 219, 221, 222, 224, 226, 228, 229, 231, 232, 233, 234, 237, 238, 239, 242, 243, 244, 245, 247, 249, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 271, 272, 273, 274, 276, 277, 278, 279, 280, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 332, 333, 334, 335, 336, 337, 338, 339, 341, 342, 343, 344, 345, 346
Validation	2, 4, 6, 8, 10, 15, 18, 20, 22, 27, 33, 35, 38, 40, 42, 46, 48, 50, 52, 54, 57, 64, 68, 69, 71, 78, 80, 85, 97, 102, 105, 107, 110, 111, 115, 118, 123, 125, 126, 128, 132, 137, 139, 143, 149, 157, 167, 169, 170, 177, 184, 186, 190, 191, 193, 198, 202, 205, 208, 220, 227, 235, 241, 248
Test	3, 5, 9, 11, 14, 17, 19, 23, 28, 29, 34, 36, 41, 43, 44, 47, 49, 53, 55, 56, 62, 67, 70, 74, 76, 79, 83, 100, 103, 104, 106, 108, 112, 116, 117, 122, 124, 127, 129, 131, 133, 138, 145, 150, 156, 161, 168, 175, 178, 181, 185, 188, 192, 194, 196, 199, 203, 209, 223, 225, 230, 236, 240, 246, 250

Table 6.1. The protocol to split the CMU MultiPIE dataset into training, validation and testing datasets.



(a) 04_1 (b) 05_0 (c) 05_1 (d) 14_0 (e) 13_0



(f) 05_1

Figure 6.2. (a-e) Illustration of the five frontal and quasi-frontal face poses from the CMU MultiPIE dataset used for training the facial feature localization models. (f) Enlarged frontal pose to better illustrate the 16 annotated facial features displayed with blue crosses.

starting at the 20×24 resolution ($P = 2$).

Given that we want to predict the location of F facial features the model has $O = 2F$ outputs, two for each facial feature consisting of the horizontal and the vertical coordinates. We use the notations $\{\mathbf{y}_{2i}, \mathbf{y}_{2i+1}\}$ and $\{\mathbf{f}_{2i}, \mathbf{f}_{2i+1}\}$ to denote the ground truth coordinates of the sample $\mathbf{y}$ and the predicted coordinates using the multivariate model $\mathbf{f}$ respectively, for the facial feature $i \in \{1, ..., F\}$. The ground truth distance between the eyes is denoted as $\Delta_{\mathbf{y}}$. The base loss and the error functions are both set to the normalized point-to-point localization error (see Table 6.2).

The error function is used to bootstrap the mis-predicted samples and to tune, on the validation dataset, the number of rounds and the regularization factor λ of the variational loss formulation. The samples were considered to have a single type ($K = 1$).

For these experiments we have trained several models to examine the impact on performance of the proposed bootstrapping method (**BOOT** vs. **SHOT**), feature sharing (**SHARED** vs. **INDEP**) and variational loss formulation (**VAR** vs. **EPT**). The parameters of all the evaluated models are presented in Table 6.2.

6.2.2 Testing protocol

We present facial feature localization results on two test datasets widely used to evaluate facial feature localization models. The first is XM2VTS (Messer et al., 1999) that contains 2360 indoor images with a single large face annotated with 68 facial feature points [1]. The second dataset is BioID (Jesorsky et al., 2001) that contains 1520 images with a single face per image annotated with 20 facial feature points [2]. We shall use only the 16 facial feature points that are consistent with the CMU MultiPIE annotations.

The test images were pre-processed using the 20×24 **EPT-BOOT** face detector and the settings described in the previous chapter. These face detections were then scaled to the resolution required by the localization model.

There are two localization settings that we consider. The first is **LEyes** which consists of predicting the eye centers ($F = 2, O = 4$). The second is **LMulti** which consists of predicting the 16

[1] The XM2VTS annotations are available at `http://personalpages.manchester.ac.uk/staff/timothy.f.cootes/data/xm2vts/xm2vts_markup.html`

[2] The BioID annotations are available at `http://personalpages.manchester.ac.uk/staff/timothy.f.cootes/data/bioid_points.html`

Setup	
Train + validation	80,000 + 80,000 samples
Resolution	$20 \times 24 \rightarrow 80 \times 96$ $(P = 2)$
Outputs	$O = 2\ F$
Features	EMB-LBP
Rounds	$R = 1024$
Loss function	$l(\mathbf{y}, \mathbf{f}) = \sum_{1 \leq i \leq F} d(\{\mathbf{y}_{2i}, \mathbf{y}_{2i+1}\}, \{\mathbf{f}_{2i}, \mathbf{f}_{2i+1}\})/(F\ \Delta_{\mathbf{y}})$
Error function	$\epsilon(\mathbf{z}\|\mathbf{f}) = \epsilon(\{\mathbf{x}, \mathbf{y}\}\|\mathbf{f}) = l(\mathbf{y}, \mathbf{f}(\mathbf{x}))$

Model name	**Description**
BOOT	$B = 3$ bootstrapping steps
SHOT	$B = 0$ bootstrapping steps
SHARED	Shared feature selection for all outputs
INDEP	Independent feature selection for each outputs
EPT	Expectation loss formulation
VAR	Variational loss formulation $\lambda \in \{0.0, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0\}$

Table 6.2. The upper half defines the common parameters for all models, while the lower half defines the parameters used for training the facial feature localization models to evaluate.

points annotated for CMU MultiPIE ($F = 16, O = 32$) and illustrated in Fig. 6.2 (f).

We have compared the proposed localization model with several baseline systems. The first is the average predictions of the facial feature locations (**AVG**). This model outputs constant predictions (relative to the detected face bounding box) computed as the average facial feature locations on the same training dataset. The other baseline systems are **HS-MLP** (Jesorsky et al., 2001) - a Hausdorff distance-based search method using a Multi-layer Perceptron (MLP) eye model, Constrained Local Model **CLM** (Cristinacce and Cootes, 2006) - an AAM-based model, and Boosted Regression coupled with Markov Networks **BoRMaN** (Valstar and Binefa, 2010) - an iterative method that uses local models refined using global geometric constraints modeled using a Markov Network. The **BoRMaN** model produces to our knowledge the best reported results on the BioID test dataset.

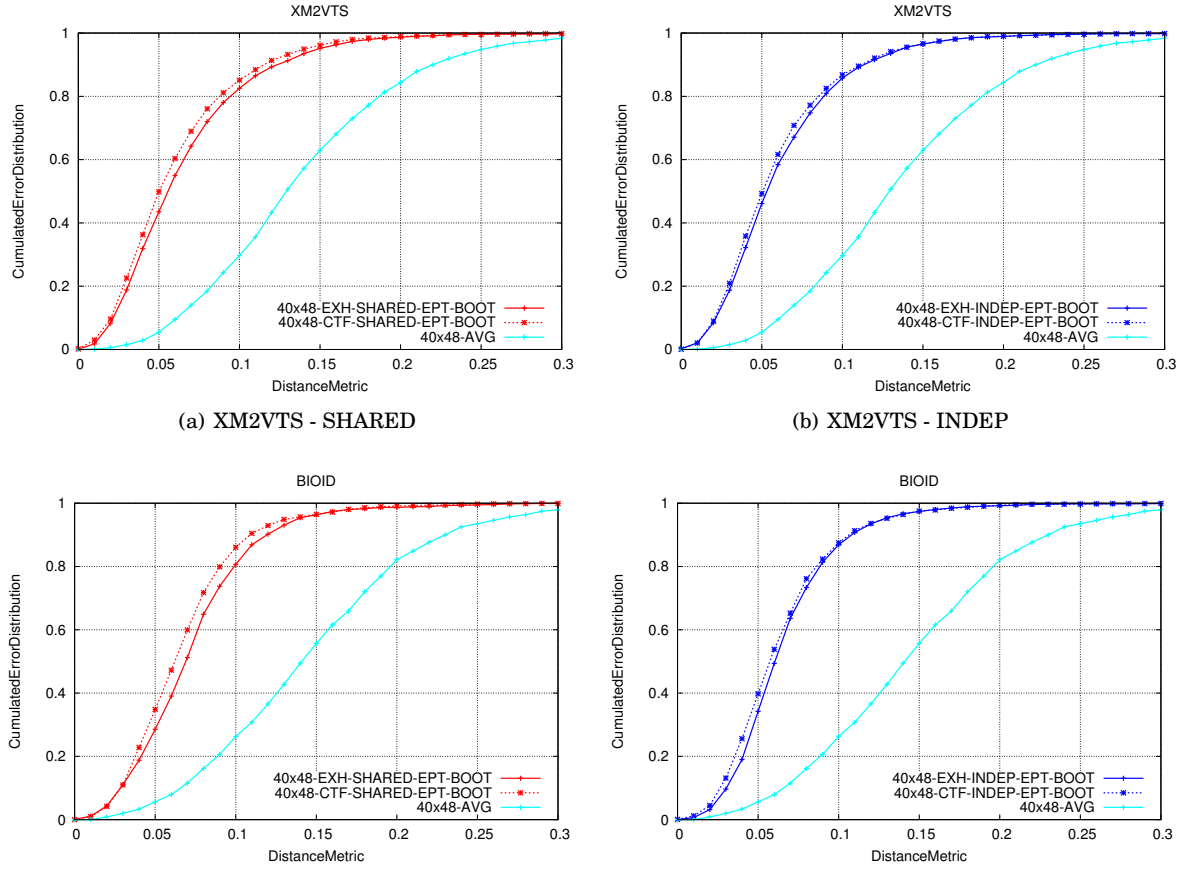
It is important to state that the training protocol for the last three systems is not clearly specified or the training datasets are not publicly available. Also, different face detectors are used as initialization. Another distinction is that we explicitly model a larger set of possible face detections, that cover at least 60% of the ground truth location. Thus, our training dataset is significantly more challenging and results in significantly less accurate average predictions produced using our training dataset (**AVG**) and the training dataset used in **CLM** (Cristinacce and Cootes, 2006).

6.3 Results and discussions

In this section we discuss the proposed coarse-to-fine feature selection method to speed-up boosting high resolution models. Next we present and discuss the experimental results obtained with models trained with different bootstrapping, feature sharing and loss parametrizations. Finally, we analyze the selected EMB-LBP feature encoding.

6.3.1 Coarse-to-fine feature selection

The first set of experiments consists of evaluating the proposed coarse-to-fine feature selection (CTFFS). We compare boosting the exhaustive set of EMB-LBP features (**EXH**) and the coarse-to-fine feature selection initialized with a 20×24 model (**CTF**). For this we have trained models of size 40×48 using the LEyes setting. These models are of lower resolution than the setup discussed above because it is not feasible to boost the exhaustive set of features for the 80×96 models. How-



(a) XM2VTS - SHARED

(b) XM2VTS - INDEP

(c) BIOID - SHARED

(d) BIOID - INDEP

Figure 6.3. The cumulated error distribution for the XM2VTS (a, b) and the BIOID (c, d) datasets using the LEyes setting. The models were trained either using shared (a, c) or independent (b, d) features between outputs.

ever, we are confident that the conclusions drawn using the 40×48 models are also valid for the 80×96 models.

The results are illustrated in Fig. 6.3. It can be noticed that CTFFS does not degrade the performance, but it actually generalizes better while processing significantly fewer features (see Table 6.3). This is for both test datasets and independently of the feature sharing method. The significant performance increase may be due to the high feature redundancy (as features per pixel), whose impact is reduced with CTFFS. The high feature redundancy may lead to boosting features that are too specific to the training dataset. In the light of these findings, we shall use implicitly coarse-to-fine feature selection with higher resolution models (80×96) for the next set of experiments.

Model	SHARED	INDEP
EXH	355,600	355,600
CTF	19,100 + 2,000	19,100 + 9,400
CTFFS speed-up	**16.8**	**12.5**

Table 6.3. The number of features to boost for the exhaustive case (**EXH**) and the coarse-to-fine feature selection case (**CTF**). The last row presents the training speed-up using **CTF** compared to **EXH**.

Model	0.05 (2px)	0.10 (4px)	0.15 (6px)	0.20 (8px)
AVG	6.1%	30.6%	61.6%	85.2%
SHARED-EPT-BOOT	39.6%	87.8%	97.1%	99.1%
SHARED-EPT-SHOT	38.3%	87.4%	96.6%	98.6%
SHARED-VAR-BOOT	38.9%	89.4%	97.6%	99.6%
INDEP-EPT-BOOT	42.9%	88.2%	97.8%	99.6%
INDEP-EPT-SHOT	47.1%	89.4%	97.5%	99.4%
INDEP-VAR-BOOT	38.3%	87.4%	97.1%	99.5%
HS-MLP (Jesorsky et al., 2001)	40%	80%	85%	87%

Table 6.4. The percentage of samples having the localization error smaller than the given threshold for various models evaluated on the BIOID dataset and the LEyes setting.

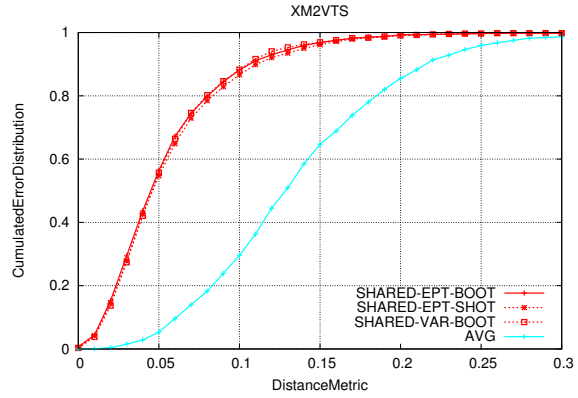
6.3.2 Performance analysis

The next set of experiments evaluated the models trained to predict the eye center locations (LEyes) and 16 facial features points (LMulti). The cumulative localization histograms are illustrated in Fig. 6.4 and Fig. 6.5 respectively. The percentage of test samples having fixed localization precision $(0.05, 0.10, 0.15, 0.20)$ are presented in Table 6.4 and Table 6.5 respectively.

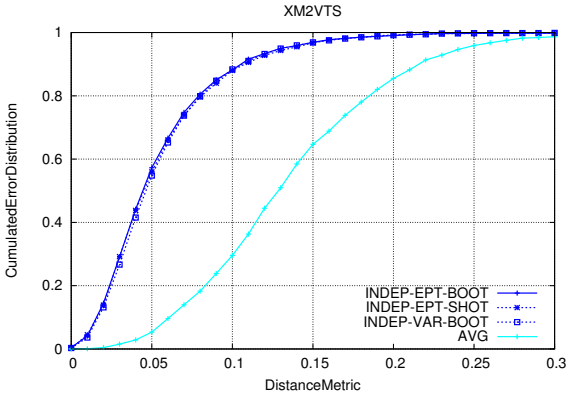
Below we highlight three important results and some example images are provided in Fig. 6.6.

Model	0.05 (2px)	0.10 (4px)	0.15 (6px)	0.20 (8px)
AVG	0.9%	13.7%	42.0%	69.3%
SHARED-EPT-SHOT	38.1%	85.5%	96.8%	98.6%
INDEP-EPT-SHOT	47.1%	88.6%	97.1%	99.3%
CLM (Cristinacce and Cootes, 2006)	45%	92%	97%	-
BoRMaN (Valstar and Binefa, 2010)	77%	95%	97%	-

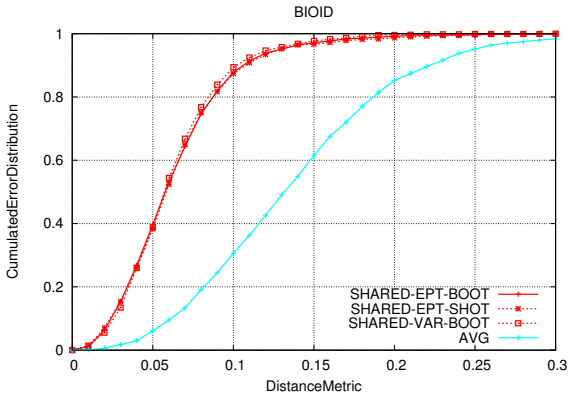
Table 6.5. The percentage of samples having the localization error smaller than the given threshold for various models evaluated on the BIOID dataset and the LMulti setting.



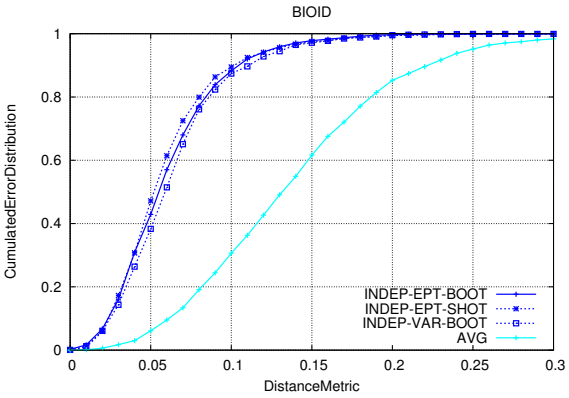
(a) XM2VTS-SHARED



(b) XM2VTS-INDEP

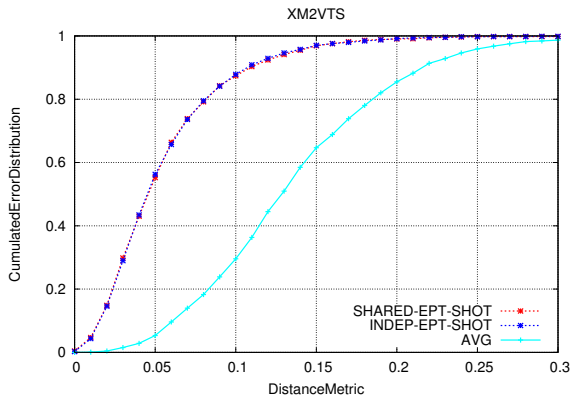


(c) BIOID-SHARED

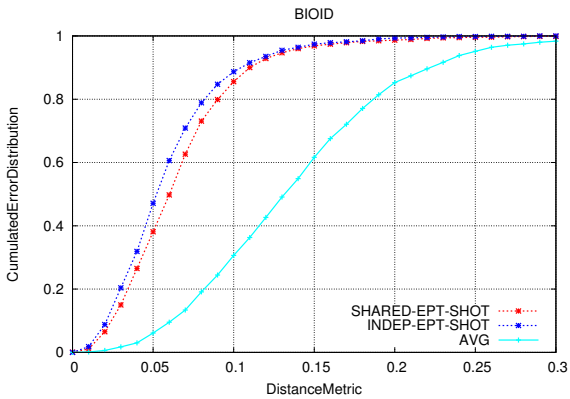


(d) BIOID-INDEP

Figure 6.4. The cumulated error distribution for the XM2VTS (a, b) and the BIOID (c, d) datasets using the LEyes setting. The models were trained either using shared (a, c) or independent (b, d) features between outputs.



(a) XM2VTS



(b) BIOID

Figure 6.5. The cumulated error distribution for the XM2VTS (a) and the BIOID (b) datasets using the LMulti setting. The models were trained either using shared or independent features between outputs.

First, the eye localization models (LEyes) are slightly more precise than the multiple feature localization models (LMulti). This is expected because the eyes are generally easier to locate and also they vary less (between people) compared to the nose tip for example. We consider that the small degradation in performance when predicting 16 points (LMulti) compared to predicting just the eye locations (LEyes) is mainly due to the loss function that jointly models the predictions for all outputs.

Second, sharing features between outputs decreases the accuracy of the model. The relative performance of the model degrades by 9% at 0.05 localization precision and by 1% at approximately 0.10 precision for the LMulti setting. However, the **SHARED** model is significantly faster to evaluate than the **INDEP** model, because it uses approximately 5 times (LEyes) and 41 times (LMulti) fewer features respectively.

Third, bootstrapping (**BOOT**) and the variational loss formulation (**VAR**) do not improve the performance of our models. This suggests that the training data is representative and clean. Randomly sampling $80,000$ training samples is representative of the whole MultiPIE training pool and bootstrapping can even degrade the performance in some cases. Also, the samples present small noise and thus using a noise-robust loss (like the variational formulation) does not improve the performance significantly.

The proposed boosted model is less accurate than state-of-the-art facial localization systems like **CLM** and **BoRMaN**. These systems are more accurate in predicting 17 facial features than the boosted model in predicting 16 facial features (LMulti setting), however, there are two key aspects that differentiate the baselines from our model. The first is that the baselines use an iterative procedure to improve the predictions at each step. The second is that the baselines use global geometric constraints on the local appearance models. Thus, future work should investigate using such geometric constraints within a boosted model. One possible way of doing this would be to add a regularization term to the cumulative loss that penalizes predictions that are not geometrically valid.

6.3.3 Feature selection analysis

We have analyzed the selected multi-block features for both localization settings to assert the type of the most useful LBP encoding type (LBP, tLBP, dLBP, mLBP) and the number of features actually

used by the shared and the independent models.

The distribution of the selected LBP encodings is more skewed than the one found for face detection or reported in literature for other tasks (Trefny and Matas, 2010). The LBP and the dLBP encodings form less than 1% of the selected features, while the mLBP accounts for 5%. The remaining tLBP encoding is used by approximately 95% of the LUTs and it is thus the most useful encoding for facial feature localization. This result is consistent across all different models and all localization settings (LEyes and LMulti) and it is also independent of the bootstrapping parameters, the loss formulation and the feature sharing method.

It is expected that sharing features between outputs uses fewer features and so produces much faster models. The speed-up is proportional to the number of outputs, but might vary because the same feature might be selected multiple times during boosting. We found experimentally that the **SHARED** models uses 5 times and 41 times fewer features than the **INDEP** models using the same parameters. This is a significant result because the **SHARED** models do not significantly reduce accuracy and at the same time are up to an order of magnitude faster.

6.4 Summary and concluding remarks

This chapter studied the proposed boosting framework on the task of facial feature localization. We have examined the impact of several aspects of the proposed model including feature sharing, bootstrapping and the variational loss formulation. Overall, we have obtained a fast facial feature localization model with good precision for practical applications, even though it does not perform as well as state-of-the-art.

The experimental findings can be summarized as:

1. Coarse-to-fine feature selection greatly speeds-up the training of high resolution models. Our experiments have shown that a multivariate 40×48 model is up to 16 times faster to train using coarse-to-fine feature selection than using the exhaustive set of features. Another empirical advantage besides speed is that the boosted model generalizes better, because fewer features are evaluated and thus the feature redundancy is lower.

2. Sharing features between outputs does not decrease the performance significantly compared with models trained with independent feature. The **SHARED** models are 5 times (LEyes) and

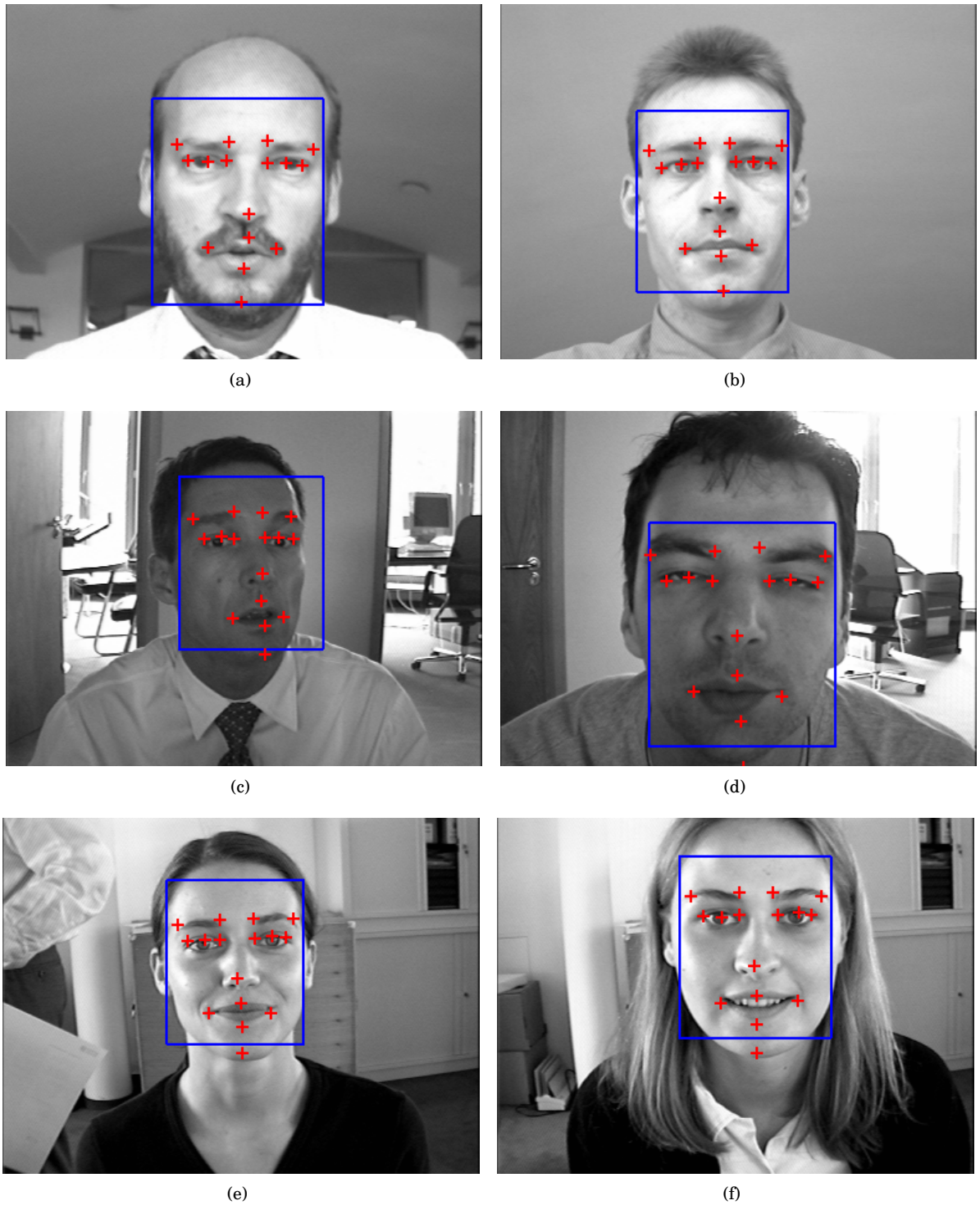


Figure 6.6. Illustration of some facial feature localization results on the BioID dataset using the LMulti setting. The face detections are represented with blue boxes and the predicted facial feature points with red crosses.

41 times (LMulti) faster to evaluate, which makes them more suitable for real-time systems. Although in this thesis we have evaluated a simple feature sharing method, there may be some better alternative which results in more accurate models.

3. Overall, bootstrapping and variational loss formulation were not found to improve performance. The slight performance increase that these methods do provide is not justified by the increasing training time. This is because the bootstrapping repeatedly evaluates the model on all the samples in the training pool and the regularization factor λ is tuned in the case of the variational formulation.

4. The transitional LBP (tLBP) encoding is selected in approximately 95% of look-up tables. This result is consistent across all localization settings and boosting configurations. The reason for this very skewed distribution of the selected LBP encoding remains unclear.

5. There are several advantages of the proposed facial feature localization model. First, the model is trained with all possible face detections and it does not need an unknown good initialization. Second, the evaluation criteria (see Eq. 6.1) is used explicitly as the training loss. However, the proposed model is very simple and it does not take into account the dependency between the facial features of interest. The experiments have shown that our model is significantly better than predicting the average location, but future work should concentrate on outperforming state-of-the-art.

The next chapter describes the experimental results for the face pose classification task.

Chapter 7

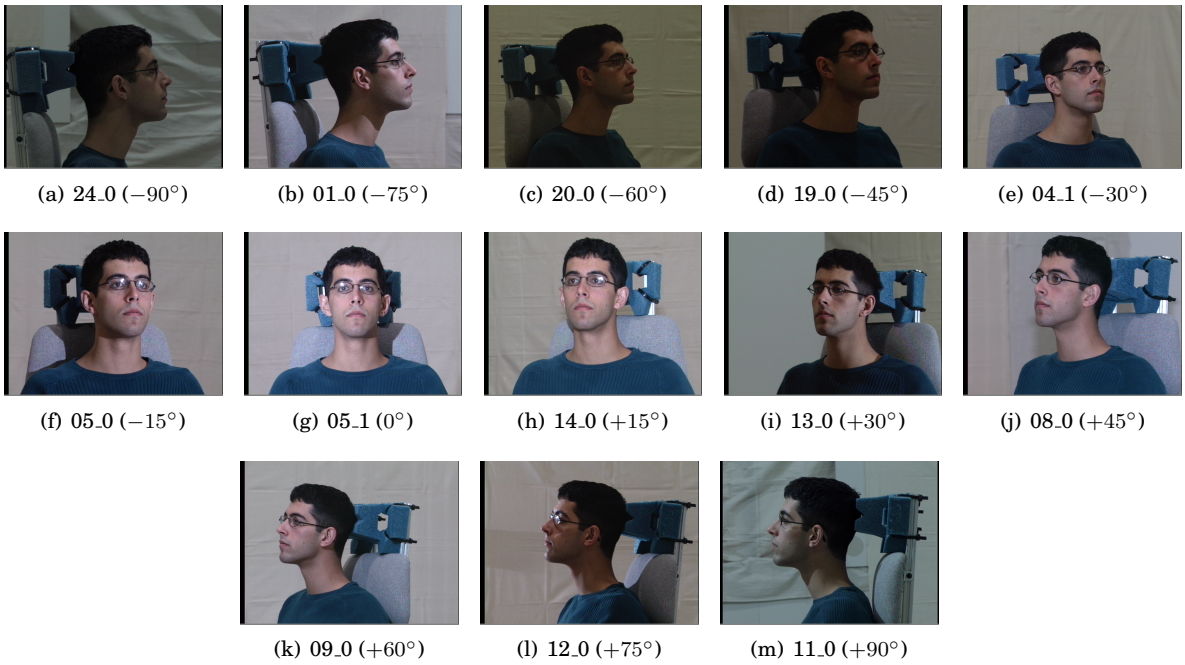
Application to face pose classification

In this chapter we apply our boosting framework to the task of pose classification. Pose classification can be used to estimate the gaze direction and the visual focus of attention of the person of the interest. This information is crucial for high-end applications for example like meeting analysis, tracking and surveillance. In this thesis we propose to estimate the face pose by classifying it into discretized out-of-plane rotations. The task is thus cast as a multi-class classification problem.

7.1 Background

The pose classification task consists of labelling the unknown pose (out of a fixed set of available labels). This is a multivariate classification problem, that we address using a model that has as many outputs as distinct poses to recognize. Ideally, the output associated with the correct label predicts a positive value, while the other outputs are negative. This is a direct multivariate generalization of the binary classification model used for face detection.

The model is evaluated using the **average error rate** and the **confusion matrix**. The average error rate is defined as the number of pose mis-predictions normalized to the total number of tested poses. More detailed information about the accuracy of the model is obtained using the confusion matrix. This is widely used in multi-class classification problems, where it is important to know



(a) 24_0 ($-90°$) (b) 01_0 ($-75°$) (c) 20_0 ($-60°$) (d) 19_0 ($-45°$) (e) 04_1 ($-30°$)

(f) 05_0 ($-15°$) (g) 05_1 ($0°$) (h) 14_0 ($+15°$) (i) 13_0 ($+30°$) (j) 08_0 ($+45°$)

(k) 09_0 ($+60°$) (l) 12_0 ($+75°$) (m) 11_0 ($+90°$)

Figure 7.1. Illustration of the 13 face poses to classify: from right profile (a) to left profile (m). We include the pose annotation from the CMU MultiPIE dataset and in brackets the degree of out-of-plane rotation.

which classes are harder to distinguish from one another. Each element in the matrix is the percentage of samples of a particular class (indexed by row) to be classified as another class (indexed by column). The ideal confusion matrix is diagonal, thus no mis-predictions are recorded. High values outside the diagonal indicates two classes that are often confused with one other, either because they are too similar or the model is not strong enough.

7.2 Experimental protocol

The experimental protocol is the same introduced in the previous section for the CMU MultiPIE dataset (see Table 6.1). We shall boost 20×24 models to classify the face pose out of 13 possible labels ($O = 13$) as illustrated in Fig. 7.1.

The training and validation datasets consists of sub-windows that overlap at least 80% with the ground truth face location. The pool of the training and validation samples is built using a fine discretization of location and scale, similar to the one used to generate samples for the face detection case. The model is trained using 1024 boosting rounds ($R = 1024$) and 7 bootstrapping steps ($B = 7$).

Setup	
Train + validation	80,000 + 80,000 samples
Resolution	20×24 $(P = 0)$
Outputs	$O = 13$
Features	EMB-LBP
Rounds	$R = 1024$
Loss function	$l(\mathbf{y}, \mathbf{f}) = \sum_{1 \leq o \leq O} log(1 + exp(-\mathbf{y}_o \mathbf{f}_o))$
Error function	$\epsilon(\mathbf{z}\|\mathbf{f}) = \epsilon(\{\mathbf{x}, \mathbf{y}\}\|\mathbf{f}) = \sum_{1 \leq o \leq O} \mathbb{1}(\mathbf{y}_o \mathbf{f}_o(\mathbf{x}) \leq 0)$

Model name	**Description**
BOOT	$B = 7$ bootstrapping steps
SHOT	$B = 0$ bootstrapping steps
SHARED	Shared feature selection for all outputs
INDEP	Independent feature selection for each outputs
EPT	Expectation loss formulation
VAR	Variational loss formulation $\lambda \in \{0.0, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0\}$

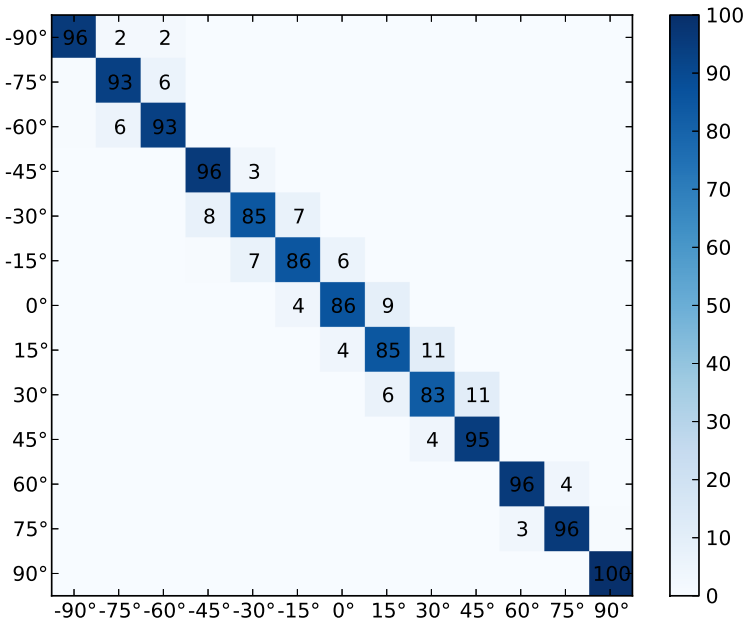
Table 7.1. The upper half defines the common parameters for all models, while the lower half defines the parameters used for training the facial feature localization models to evaluate.

We sample 80,000 training samples and 80,000 validation samples ($N_0^t = 10,000, N_0^v = 80,000$). The samples were considered to have 13 different types ($K = 13$).

We have trained several models to assert the performance impact of the proposed bootstrapping method (**BOOT** vs. **SHOT**), feature sharing (**SHARED** vs. **INDEP**) and variational loss formulation (**VAR** vs. **EPT**). The parameters of all the evaluated models are presented in Table 7.1.

Model	Average error rate
INDEP-EPT-BOOT	8.51%
INDEP-EPT-SHOT	8.61%
INDEP-VAR-BOOT	9.07%
SHARED-EPT-BOOT	12.18%
SHARED-EPT-SHOT	12.31%

Table 7.2. The pose classification average error rate for the boosted models on the CMU MultiPIE test dataset.



(a)

Figure 7.2. The confusion matrix for the **INDEP-EPT-BOOT** models on the CMU MultiPIE test dataset. The poses are arranged from right to left profile, such that the frontal pose is in the middle.

7.3 Results and discussions

The CMU MultiPIE test dataset is used to evaluate the boosted models. No baseline is available for this task and so we shall only compare variations of our proposed approach. The average error rate is presented in Table 7.2 and a typical confusion matrix in Fig. 7.2.

7.3.1 Performance analysis

The immediate conclusion is that sharing features produces models that are significantly worse (up to 50% relative error increase) than boosting independent features, consistent for all loss formulation and bootstrapping steps. This is in contrast to facial feature localization where sharing features does not decrease performance significantly. This suggests that our current method of performing feature sharing is task dependent.

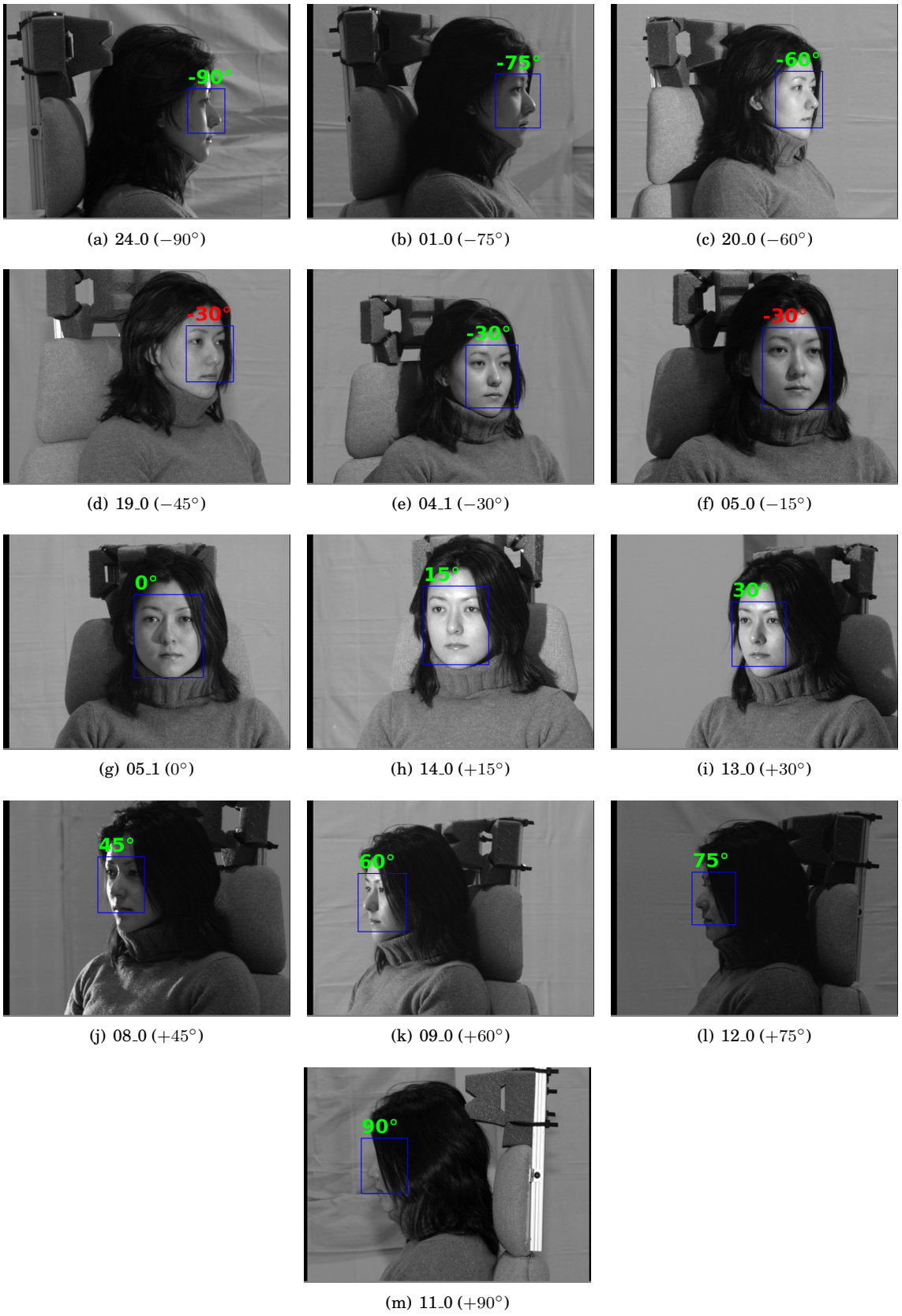
It can also be noticed that bootstrapping slightly increases the performance (up to an 0.2% error decrease) for both feature sharing methods. Bootstrapping increases the training time by no more than 50% which is acceptable, however, the variational loss formulation actually decreases the performance by 0.5%. Also, this formulation is very time consuming because the regularization term λ needs to be tuned on the validation dataset. This suggests that the variational loss formulation is not useful for this task.

The confusion matrix shows that errors are distributed close to the diagonal. This suggests that the predictions are accurate, but also that vast majority of mis-predictions involve the adjacent (out-of-plane) poses. Considering that there are 13 poses evenly distributed over 180 degrees of in-plane rotation, then each pose covers 15 degrees. If we look only at adjacent poses we obtain 1% error rate. This accuracy is clearly sufficient for real-world applications.

An interesting finding is that the accuracy decreases for near frontal poses. This is easily observed by analyzing the diagonal results in Fig. 7.2. It can be seen that for the poses between -30 degrees and +30 degrees are much more easily confused with one another; this only occurs with the adjacent class, for instance confusability between 0 degrees and -15/+15 degrees. This suggests that it is much more difficult to predict near frontal (-30 to +30 degrees) than close to profile poses. We believe that this is because of the larger inter-person variations that occur with poses beyond -30/+30 degrees, some results are illustrated in Fig. 7.3 and Fig 7.4.

7.3.2 Feature selection analysis

The feature selection analysis shows the same pattern: the tLBP (transitional LBP) encoding is selected significantly more frequent that the other LBP encoding (mLBP, dLBP, LBP). The **INDEPEPT-BOOT** model consists of look-up tables with 70.8% tLBP and 23.4% mLBP features respec-



(a) 24_0 (−90°) (b) 01_0 (−75°) (c) 20_0 (−60°)

(d) 19_0 (−45°) (e) 04_1 (−30°) (f) 05_0 (−15°)

(g) 05_1 (0°) (h) 14_0 (+15°) (i) 13_0 (+30°)

(j) 08_0 (+45°) (k) 09_0 (+60°) (l) 12_0 (+75°)

(m) 11_0 (+90°)

Figure 7.3. Illustration of some face pose classification results on the CMU MultiPIE test dataset: from right profile (a) to left profile (m). The face bounding box is represented with the blue rectangle. The classified poses are displayed in angles with green if correct and with red if incorect respectively.

(a) 24_0 (−90°) (b) 01_0 (−75°) (c) 20_0 (−60°)

(d) 19_0 (−45°) (e) 04_1 (−30°) (f) 05_0 (−15°)

(g) 05_1 (0°) (h) 14_0 (+15°) (i) 13_0 (+30°)

(j) 08_0 (+45°) (k) 09_0 (+60°) (l) 12_0 (+75°)

(m) 11_0 (+90°)

Figure 7.4. Illustration of some face pose classification results on the CMU MultiPIE test dataset: from right profile (a) to left profile (m). The face bounding box is represented with the blue rectangle. The classified poses are displayed in angles with green if correct and with red if incorect respectively.

tively. The associated feature sharing model **SHARED-EPT-BOOT** presents an even more skewed feature distribution: 94.3% tLBP and 4.6% mLBP. Once more, the tLBP feature is the most useful encoding for pose classification too. This result is consistent across the bootstrapping parameters, the loss formulation and the feature sharing method.

7.4 Summary and concluding remarks

This chapter presented the results obtained with the proposed boosting approach on the pose classification task. The goal was to predict the correct pose out of 13 available poses. We found experimentally that the best boosted model produces predictions of around 8% and 1% average error rate for 15 degrees and 30 degrees precision, respectively. This has shown that our boosting framework can be effectively applied to a multivariate classification task such as pose classification.

Chapter 8

Conclusions and future work

The standard approach to face processing involves a mixture of machine learning models and features. This is motivated by the diversity of the face processing tasks including: face detection, facial feature localization, face recognition and pose classification. These tasks are often connected in a processing chain: first detection, then alignment (or localization), then recognition or other high level tasks. This may be inefficient because these models often require different pre-processing steps and features that cannot be shared across the tasks.

In this thesis we propose a generic multivariate boosting framework to address several face processing tasks. This is possible because model training is performed as optimizing a generic loss. In particular, we propose to boost look-up tables with local binary features motivated by their evaluation and computation speed, respectively, and their proven robustness on similar tasks. These models can be connected into efficient and homogeneous face processing chains.

8.1 Experimental findings

The proposed multivariate boosting framework was applied to several face processing tasks: face detection, facial feature localization and pose classification. Each task was discussed in detail using appropriate experimental protocols and baselines (if available). The same boosting procedure and the same features (EMB-LBP) were used for all tasks.

This approach presents several important advantages:

1. Modeling is easier and it reduces to formulating a task-appropriate loss along with some regularization terms.

2. Boosting look-up tables is shown to be fast and scalable with the available resources at training time. The combination of boosted LUTs and LBP features results in a fast and robust system.

3. The approach is generic. It is suitable for a large variety of classification and regression problems, with single or multiple outputs. In addition to this, given a base loss that associates a value (error) to predictions, we formulate both the expectation and the variational losses. The variational formulation has the advantage of being more robust to noise regardless of the base loss or the task at hand.

The experiments were designed to assess the influence of various aspects of the boosting framework including: bootstrapping the training samples, using loss formulations (expectation and variational), using feature sharing and using coarse-to-fine feature selection. Overall, the boosted models achieve state-of-the-art on face detection and perform reliably on facial feature localization and pose classification.

We list the experimental findings below.

1. The proposed ad-hoc splitting of the boosted model into a cascade of levels (stages) produced a face detector that achieves state-of-the-art performance with 7.12 weak learner evaluations per sub-window on average. This is the fastest face detector reported in literature.

2. Sharing features between outputs produces much faster models, for example of about 5 to 41 times faster for the facial feature localization task. This comes with some performance degradation, that is insignificant for facial feature localization but unacceptable for pose classification. We conclude that the trade-off between speed (how many outputs to share a feature) and performance is task specific.

3. The variational loss formulation is usually more robust than the expectation formulation. Significant performance improvements are found for face detection and pose classification, while only minor improvements for facial feature localization. However, the variational models are

more expensive to train, because the regularization factor is task specific and it needs to be tuned on the validation dataset.

4. The proposed coarse-to-fine feature selection is found to be an efficient method for boosting any multi-block features for high resolution models. The resulting models are many times faster to train (than considering the exhaustive set of features) and also more robust mostly because the feature redundancy (number of features per pixel) is greatly reduced.

5. Bootstrapping improves the performance for face detection and pose classification tasks at a moderate increase in the training time. However, the bootstrapped models are less accurate for facial feature localization. Further improvements are suggested in the following section.

6. The transitional LBP (tLBP) encoding was found consistently to be the most frequently selected out of the four LBP encoding schemes combined in the EMB-LBP feature. The selection percentage reaches 95% for facial feature localization irrespective of the boosting settings. The very skewed LBP encoding distribution contradicts the findings reported in the original paper that introduced EMB-LBP (Trefny and Matas, 2010), albeit for different pattern recognition problems.

8.2 Directions for future work

The following are some general work directions relevant to boosting and to face processing.

1. The proposed boosting formulation does not take into account the dependencies between outputs. For example, in the case of the facial feature localization task, the mouth corner location is clearly dependent on the eye location. More generally, we consider that modeling geometric constraints between predictions (as a regularization term for example) may improve the performance of the boosted model.

2. Optimal feature sharing is of great interest in pattern recognition. Though the feature sharing methods discussed in this thesis are very simple, we consider that more sophisticated and more reliable methods can be integrated within the proposed generic boosting framework.

3. Boosting multivariate models is expensive, although scalable with the available resources. It is of interest to assess the trade-off between the number of bootstrapping steps and the number of training samples. We consider that it is possible to boost models of similar performance with significantly fewer training samples (thus faster), by performing more bootstrapping steps with slower increase in the number of rounds. The optimal bootstrapping procedure remains an open problem.

4. The proposed approach is generic. Hence, it could be extended to other face processing tasks, for example face recognition and face verification applications. It is of interest to also study the selected LBP encodings (as part of the EMB-LBP feature) for these applications.

Appendices

Appendix A

Face detection using boosted Jaccard distance-based regression

This appendix presents a new face detection method. We train a model that predicts the Jaccard distance between a sample sub-window and the ground truth face location. This model produces continuous outputs as opposed to the binary output produced by the widely used boosted cascade classifiers. To train this model we introduce a generalization of the binary classification boosting algorithms in which arbitrary smooth loss functions can be optimized. This way single output regression and binary classification models can be trained with the same procedure.

Our method presents several significant advantages. First, it circumvents the need for a specific discretization of the location and scale during testing. Second, it provides an approximation of the search direction (in location and scale) towards the nearest ground truth location. And finally, the training set consists of more diverse samples (e.g. samples covering portions of the faces) that cannot be used to train a classifier. We provide experimental results on the BioID face dataset to compare our method with the sliding-windows approach.

A.1 Objectives and motivations

Face detection consists of finding the position of all the faces, if any, in an image. Two components are usually required: a classifier and a search algorithm. The search (or scanning) algorithm forms

sub-windows (or samples) at different locations and scales which are fed to the classifier. The sub-windows labelled as positive samples are considered as final detections. Usually a clustering algorithm (e.g. non-maxima suppression, averaging the overlapping regions, mean shift) is run on these detections to reduce the number of multiple detections.

Recently there has been a great interest in real-time face detection systems. Their speed depends mostly on the speed of the classifier to evaluate a sub-window. These systems are usually built using boosted classifiers (Viola and Jones, 2004; Zhang et al., 2007), because of their potential computational efficiency while providing state of the art performance. Another important factor is the speed to compute the features. The fastest features are evaluated in constant complexity at any location and scale, for example: Haar-like features (Viola and Jones, 2004) and MCT (Froba and Ernst, 2004) or multi-block LBP codes (Zhang et al., 2007).

The most popular and simple search strategy for face detection is the sliding-windows approach (we refer to this method as ***SScan***). The location and scale space is usually discretized using a fixed grid or a coarse-to-fine approach.

There are two problems with the *SScan* approach that we address in this appendix. First, the discretization parameters are difficult to automatically adjust to the size of the image to scan and it clearly depends on the (unknown) number, size and distance between adjacent faces. Second, the classifiers cannot be trained with samples that cover just a part of the face. For example it is impossible to decide if a sample containing just half of the face should be considered as a positive or as a negative training sample. Therefore an uncertain region around the ground truth is formed during training (Everingham and Zisserman, 2006; Cristinacce and Cootes, 2007). But these kind of samples consistently appear at testing time and there is no guarantee on the classifier's output in this situation.

A possible solution is to use regression to learn a richer information than just a label of a sub-window to test. There has been some previous work on this research direction. For example a boosted model is trained in (Cristinacce and Cootes, 2007) to predict the displacement of a facial feature patch from the ground truth. In (Everingham and Zisserman, 2006) a regression approach is used to predict the eye positions. Our work follows this direction.

More specifically, we propose a new real-time face detection method that uses regression to guide the search. We train a model that predicts the Jaccard distance between a sample sub-window and

the nearest ground truth face location. Then this model is used to search for faces in two steps. First we initialize a set of potential detections with a coarse sampling and second we iteratively refine the most promising detections. We refer to this method as ***JScan***.

The main contributions of this work can be summarized as follows:

- We train a model to learn how accurate a sub-window is in both location and scale. This allows for arbitrary displacements and scale variations of the ground truth face locations sub-windows in the training samples. We then use this model for face detection without the need for a specific discretization of the search space.
- We propose a general formulation of boosting algorithms that is independent of the loss function and the specific classification or regression task to solve. This formulation allows for training binary classifiers and single output regressors with the same algorithm.
- An additional contribution is the proposed features that combine Multi-Block Local Binary Patterns and Modified Census Transform features.

A.2 Related work

A.2.1 Boosting

Boosting (Schapire, 2002) is a greedy method for learning a *strong classifier* as a linear combination of *weak classifiers*. This process is done iteratively in *boosting rounds*: a single new weak classifier is chosen and added to the combination. Each new weak classifier is usually trained to correct the mistakes made by the previous ones and to focus on the most challenging samples. Boosting can also be interpreted as a gradient descent algorithm in the functional space of the weak classifiers (Mason et al., 1999a).

In this section we focus on a more general formulation of boosting as a greedy optimization of the Taylor expansion of the loss function to optimize (Mason et al., 1999a; Torralba et al., 2007). This has the advantage of having the same formulation for both classification and regression, allowing for an easy and fair comparison between classification and regression methods for face detection.

More formally let $\boldsymbol{\chi}$ be the input signal space and $\{(x_n, y_n)_{n=1:N}\} \in (\boldsymbol{\chi} \times \mathbb{R})^N$ a set of N training samples. The targets $\{y_n\}$ to learn can be either binary labels $\{-1, +1\}$ or any other scalar for

regression problems. The goal is to build a functional $f : \chi \rightarrow \mathbb{R}$ to map the samples x_n to their targets y_n. At each boosting round t, $t \leq T$, its approximation is a linear combination of g_s *weak learners*: $f_t = \sum_{s \leq t} g_s$.

The criteria to choose f is to minimize a loss function of the form: $L(f_t) = \sum_n l(y_n, f_t(x_n))$. At each step $t+1$ the weak learner g_{t+1} is chosen to minimize:

$$g_{t+1} = \arg\min_g \sum_n l(y_n, f_t(x_n) + g(x_n)). \tag{A.1}$$

Taking the Taylor expansion of the loss, in the current f_t point, up to the second order, the optimization problem becomes:

$$\begin{aligned} g_{t+1} = & \arg\min_g L(f_t) + \\ & \sum_n \left(\frac{\partial l(y_n, f)}{\partial f} \mid_{f=f_t(x_n)} \right) g(x_n) + \\ & \frac{1}{2} \sum_n \left(\frac{\partial^2 l(y_n, f)}{\partial f^2} \mid_{f=f_t(x_n)} \right) g^2(x_n). \end{aligned} \tag{A.2}$$

Most boosting algorithms can be derived from this formulation. We distinguish between second order and first order boosting algorithms depending if they use or not the second order term in the Taylor expansion in Eq. A.2.

The first order boosting algorithms perform a gradient descent in the functional space:

$$g_{t+1} = \arg\max_g |\sum_n \left(\frac{\partial l(y_n, f)}{\partial f} \mid_{f=f_t(x_n)} \right) g(x_n)|. \tag{A.3}$$

For example: AdaBoost (Schapire, 2002) minimizes the exponential loss $l(y, f) = exp(-yf)$ and restricts the weak classifiers to the form $g_s : \chi \rightarrow \{-1, +1\}$, while "AnyBoost" (Mason et al., 1999a) is a generic formulation for any loss functions. It can be noticed that the optimal weak learner g_{t+1} is known up to a scaling factor. This is the reason why g_{t+1} is typically scaled using the line-search algorithm, fixed steps or decreasing steps. In particular cases the optimal scale can be found analytically (e.g. AdaBoost).

The second order boosting algorithms use adaptive Newton steps to minimize the loss function. A well known algorithm of this type is Gentle AdaBoost (Friedman et al., 2000; Torralba et al.,

Classification:	$l_1(y,f) = exp(-yf)$
	$l_2(y,f) = log(1+exp(-yf))$
Regression:	$l_3(y,f) = \frac{1}{2}(y-f)^2$
	$l_4(y,f) = exp(y-f) + exp(f-y) - 2$

Table A.1. Various loss functions for classification (l_1, l_2) and regression (l_3, l_4).

2007) which optimizes the exponential loss for the classification task.

Usually the normalized partial derivatives of the loss function are considered as *weights* associated to each sample (e.g. AdaBoost, Gentle AdaBoost). This allows, in certain conditions for the classification task, to interpret boosting as a greedy algorithm that concentrates the current weak learner on the samples mis-classified by the previous weak learners.

The loss function depends on the specific problem to solve (see Table A.1). For example, the classification task requires the two classes to be separated as far as possible: $l(y,f) = l(-yf)$, while the regression task needs a prediction as close as possible to the target: $l(y,f) = l(y-f)$.

A.2.2 Face detection using sliding-windows (SScan)

The Algorithm 8 presents the sliding-windows approach to face detection. Given a face classifier M that processes sub-windows of size $M^w \times M^h$, the algorithm searches for faces in the image I of size $I^w \times I^h$. The discretization of the location and scale space is governed by the dx, dy and ds parameters. The dx and dy parameters are used to compute the displacement in location between two sub-windows, relative to the model size, for the scaled image I_s of size $I_s^w \times I_s^h$ by the s factor. If the classifier scores above a given threshold τ, then the detection $det = \{x, y, s\}$ is accepted in the final list D.

There are several problems with this method. First, the dx, dy and ds parameters are difficult to set a priori. They dependent on the size of the image to search and the number of and the distance between face locations. Second, the search algorithm uses a face classifier that is trained with roughly normalized samples. For example it cannot be trained with samples that cover just a part of the face. This is because it is impossible to decide if a sample containing just half of the face should be considered as a positive or as a negative training sample. Therefore an uncertain region around the ground truth is formed during training (Everingham and Zisserman, 2006; Cristinacce

Algorithm 8 Face detection using sliding-windows.

$dx \in (0,1), dy \in (0,1), ds \in (0,1), \tau, M, I, D = \phi$
for $s = 1$ **to** $s > 0$ **do**
  scale the image: $I_s \leftarrow I \otimes s$
  for $x = 0$ **to** $x < I_s^w$ **do**
    for $y = 0$ **to** $y < I_s^h$ **do**
      $det = \{x, y, s\}$
      if $M(det) \geq \tau$ **then**
        $D \leftarrow D \cup det$
      end if
      $y \leftarrow y + dy * M^h$
    end for
    $x \leftarrow x + dx * M^w$
  end for
  $s \leftarrow s - ds$
end for
return D

and Cootes, 2007). The problem is that there is no guarantee on the output of the classifier for these kind of sub-windows that can appear during testing.

A.3 Proposed approach

In this section we introduce the features and the weak learner (A.3.1). Then we describe the training algorithm (A.3.2) using the framework presented in the previous section. Next we present the Jaccard distance (A.3.3) and how to use it for face detection (A.3.4).

A.3.1 Features and weak learner

A real-time face detection system requires features that are fast to compute at any location and scale. The first real-time system used Haar-like features (Viola and Jones, 2002), but LBP-based features also became very popular because they are robust to illumination changes (Froba and Ernst, 2004). Recently, the Multi-Block LBP features (Zhang et al., 2007) have been shown to outperform both Haar-like features and LBP codes. Hence, in this work we use a new feature - the Multi-Block Modified Census Transform (MB-MCT), that combines the multi-block idea proposed in (Zhang et al., 2007) and the MCT features proposed in (Froba and Ernst, 2004).

The MB-MCT features are parametrized by the top-left coordinate (x, y) and the size $w \times h$ of the rectangular cells in the 3×3 neighbourhood. This gives a region of $3w \times 3h$ pixels to compute

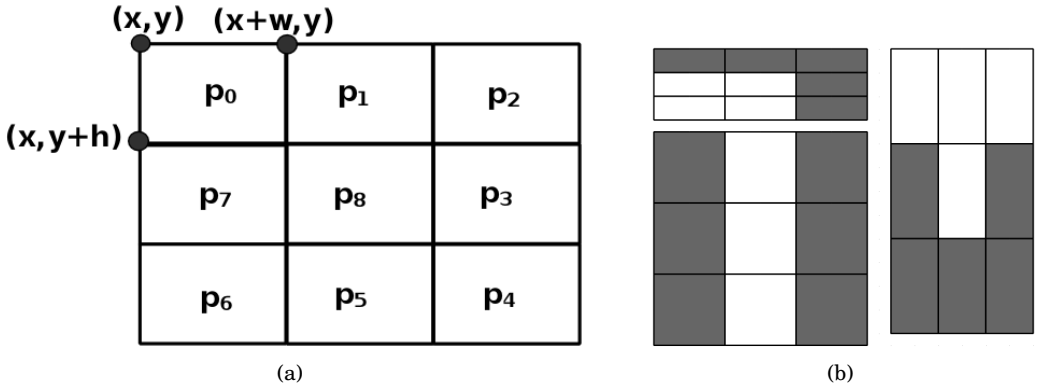


Figure A.1. (a) Multi-block MCT feature for image representation. (b) Examples of some patterns that can be obtained by varying the parameters w and h.

the 9-bit MB-MCT:

$$MB - MCT(x, y, w, h) = \sum_{i=0:8} \delta(p_i \geq \bar{p}) * 2^i, \tag{A.4}$$

where δ is the Kronecker delta function, $\bar{p}$ is the average pixel intensity in the 3×3 region and p_i is the average pixel intensity in the cell i (see Fig. A.1 (a)). The feature is computed in constant time for any parametrization using the integral image. Various patterns at multiple scales and aspect ratios can be obtained by varying the parameters w and h (see Fig. A.1 (b)).

The MB-MCT feature values are non-metric codes and this restricts the type of weak learner to boost. We use the multi-branch decision tree proposed in (Zhang et al., 2007) as weak learner. This weak learner is parametrized by a feature index (e.g. dimension in the feature space) and a set of fixed outputs, one for each distinct feature value. More formally, the weak learner g is computed for a sample x and a feature d with:

$$g(x) = lut[x^d], \tag{A.5}$$

where lut is a look-up table with 512 entries a_u (because there are 512 distinct MCT codes) and d indexes in the space of x, y, w, h possible MB-MCT parametrizations. The goal of the boosting algorithm is then to compute the optimum feature d and a_u entries.

A.3.2 Training

In this section we derive the second order boosting algorithm to train the multi-branch decision tree. We chose this formulation over the first order because generally the loss decreases faster using Newton-Raphson steps than using gradient descent steps. The Eq. A.2 can be rewritten for a fixed feature d as:

$$g_{t+1}(d, a_u) = \arg\min_{a_u} L(f_t) + \sum_u a_u \left(\sum_{n, x_n^d = u} \left(\frac{\partial l(y_n, f)}{\partial f} \mid_{f=f_t(x_n)} \right) \right) + \sum_u a_u^2 \left(\frac{1}{2} \sum_{n, x_n^d = u} \left(\frac{\partial^2 l(y_n, f)}{\partial f^2} \mid_{f=f_t(x_n)} \right) \right) \tag{A.6}$$

or more compactly as:

$$g_{t+1} = \arg\min_{d, a_u} L(f_t) + \sum_u a_u L'_u + \sum_u \frac{1}{2} a_u^2 L''_u, \tag{A.7}$$

where L'_u and L''_u are the cumulated first and second order derivatives of the loss for the samples that have the feature d with the value u. It can be noticed that the quadratic optimization problem can be solved **separately** for each look-up-table entry a_u. The exact solution and the potential loss decrease Δ are:

$$a_u = -\frac{L'_u}{L''_u} = -\frac{\sum_{n, x_n^d = u} \left(\frac{\partial l(y_n, f)}{\partial f} \mid_{f=f_t(x_n)} \right)}{\sum_{n, x_n^d = u} \left(\frac{\partial^2 l(y_n, f)}{\partial f^2} \mid_{f=f_t(x_n)} \right)} \tag{A.8}$$

$$\Delta = -\sum_u \frac{(L'_u)^2}{2 L''_u}. \tag{A.9}$$

The training algorithm is presented in Algorithm 9. At each boosting round t, the optimal weak learner g_{t+1} is chosen by evaluating each feature d and selecting the optimal one with the highest decrease Δ in the loss. Then g_{t+1} is added to the strong model f. It can be noticed that the algorithm

is of linear complexity with the number of samples, features and boosting rounds. There are two important benefits: it performs feature selection and it can be used for any smooth loss function. In this appendix we use this algorithm for both face classification and Jaccard distance-based face regression.

Algorithm 9 Second order boosting multi-branch MB-MCT decision trees.

for $t = 0, f = 0$ **to** $t \leq T$ **do**
  $d^* = 0, \Delta^* = \infty, a_u^* = 0$
  for feature d **do**
    for feature value $u \in 0...511$ **do**
      compute L'_u and L''_u
    end for
    $\Delta = -\sum_u \frac{\left(L'_u\right)^2}{2L''_u}$
    if $\Delta < \Delta^*$ **then**
      $d^* \leftarrow d, \Delta^* \leftarrow \Delta, a_u^* \leftarrow -\frac{L'_u}{L''_u}$
    end if
  end for
  $f \leftarrow f + g_{t+1}(d^*, a_u^*), t \leftarrow t + 1$
end for
return f

A.3.3 Jaccard distance

The Jaccard distance (Jaccard, 1901) is a statistical method to measure the similarity between two sets A and B (see Eq. A.10).

$$J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}. \tag{A.10}$$

This can be extended to measure the overlap between two rectangular regions. Then, $|A \cap B|$ and $|A \cup B|$ stand for the area of their intersection and union, respectively. We decided to use an approximation to the Jaccard distance that it is easier to compute in the case of face detection:

$$J_m(A, B) = 1 - \frac{|A \cap B|}{max(|A|, |B|)} \tag{A.11}$$

Let A be the ground truth face location and B a sub-window to evaluate. Then, the perfect detection corresponds to the distance $J_m(A, B) = 0$, while the background sub-windows corresponds to the distance $J_m(A, B) = 1$. The target y to learn for the particular sub-window B is $J_m(A, B)$.

A.3.4 Face detection using Jaccard distance-based regression (JScan)

We propose a regression-driven search algorithm for face detection. Instead of using a classifier, we train a model to learn a richer information: the Jaccard distance between a sub-window and the closest ground truth face location. An immediate benefit of using regression is that the model can be trained with sub-windows at any location and scale, which implies that no uncertain region is formed any more.

Assuming that such a model M is provided, the search algorithm becomes as presented in Algorithm 10. There are several significant differences compared to Algorithm 8. First, no dx, dy, ds discretization is needed any more. This is because the model predicts how far the current sub-window is from the true face location, which can be used to guide the search instead of some a priori fixed discretization parameters. Second, the proposed method is split in two stages: the initialization of potential locations (steps 8, 10 and 12) and the refinement (steps 14, 15) of these locations to minimize the Jaccard distance. It can be noticed that we refine only the sub-windows that are close to the ground truth. This has the benefit of concentrating the effort (evaluating sub-windows) in the most promising regions of the search space. Finally, we ignore the detections that are farther away than τ from the true location (step 16). This corresponds to eliminating false alarms (see Algorithm 8, step 7).

Initialization stage (steps 1-12)

The search for the optimal face locations is initialized using an uniform grid. The idea is *to sample such that half of a face is ensured to be included in some sub-window, such that* $J_m \leq 0.5$. This implies that we need to sample every $\frac{M^w}{2}$ and $\frac{M^h}{2}$ on the horizontal and vertical axis, respectively. The scale sampling factor is slightly more difficult to set. Let s be the current scale. Then a sub-window at this scale has the size of $\frac{1}{s}M^w \times \frac{1}{s}M^h$ relative to the original image. The difference in size between two sub-windows at consecutive scales s and $s' < s$ is: $(\frac{1}{s'} - \frac{1}{s})M^w \times (\frac{1}{s'} - \frac{1}{s})M^h$. To make sure that half of a face is contained in some sub-window we set the conditions: $(\frac{1}{s'} - \frac{1}{s})M^w \leq \frac{M^w}{2}$ and $(\frac{1}{s'} - \frac{1}{s})M^h \leq \frac{M^h}{2}$. This implies $\frac{1}{s'} - \frac{1}{s} \leq \frac{1}{2}$. At the limit, it can be shown that we obtain the following relation for the scale variation: $s_n = \frac{2}{n+1}, n \geq 1$.

Refinement stage (steps 14 and 15)

Next the detections close enough to the ground truth locations are *refined* in two steps. Given that the Jaccard distance is isotropic, the optimal direction where the face location resides cannot be deduced from this information. Instead we sample *independently* each axis (horizontal, vertical and scale) with two values (left - right, down - up, bigger - smaller). In the first step we sample with the half, while in the second step with the quarter of the displacement used in the initialization. The sampling spacing is decreased because smaller steps are required as the detection gets closer to the ground truth locations. Only the detections that are within 0.50 and 0.40 of the modified Jaccard distance (Equation A.11) from the ground truth are refined at the first and the second step respectively. It can be noticed that it is pointless to have more than two refinement steps because the spacing resolution (divided by two at each step) reaches the limit. For example, for the sub-window $(x, y, \frac{1}{s})$ we generate at the first refinement step the six sub-windows to refine: $(x \pm \frac{M^w}{4}, y \pm \frac{M^h}{4}, \frac{1}{s} \pm \frac{1}{4})$. Considering a model of the size 24×24, the initialization part process locations at every 12 pixels, while the refinement steps at every 6 and 3 pixels respectively.

Algorithm 10 Face detection using Jaccard distance-based regression.

1: $\tau \in (0,1), M, I, D = \phi$
2: **for** $s = 1$ **to** $s \geq max(\frac{M^w}{I^w}, \frac{M^h}{I^h})$ **do**
3: $I_s \leftarrow I \otimes s$
4: **for** $x = 0$ **to** $x < I_s^w$ **do**
5: **for** $y = 0$ **to** $y < I_s^h$ **do**
6: $det = \{x, y, s\}$
7: $D \leftarrow D \cup det$
8: $y \leftarrow y + \frac{M^h}{2}$
9: **end for**
10: $x \leftarrow x + \frac{M^w}{2}$
11: **end for**
12: $\frac{1}{s} \leftarrow \frac{1}{s} + \frac{1}{2}$
13: **end for**
14: $refine(D, 0.50, x \pm \frac{M^w}{4}, y \pm \frac{M^h}{4}, \frac{1}{s} \pm \frac{1}{4})$
15: $refine(D, 0.40, x \pm \frac{M^w}{8}, y \pm \frac{M^h}{8}, \frac{1}{s} \pm \frac{1}{8})$
16: $threshold(D, \tau)$
17: **return** D

A.4 Experiments and results

As a proof of concept, we have performed experiments to investigate the feasibility of the proposed idea. For this we have compared our proposed face detection method with the sliding-windows approach using an equivalent complex boosted classifier.

A.4.1 Experimental setup

The training samples were generated using the BANCA (Bailly-Bailliére et al., 2003) English face dataset (6240 images) and the CALTECH-101 (Fergus and Perona, 2007) background dataset (451 images). The BANCA dataset contains images taken in controlled and uncontrolled conditions of a single person in an office environment. We normalized these images to have the eyes horizontally aligned and with 32 pixels distance between them. Then we collected roughly 6 billion sub-windows of size 24×24 using a very fine discretization of location and scale.

Each model was trained using 500,000 randomly selected samples. The training samples were generated to be evenly distributed over the output values: the class labels $\{-1, +1\}$ for classification and the Jaccard distance values $[0, 1]$ for regression. The classifier required one more restriction to overcome the uncertain area problem: the positive samples had to overlap at least 90%, while the negative samples had to overlap at most 10% respectively with the ground truth face location.

The same feature parametrization (see Section A.3.1) was used for both models. The MB-MCT features were generated with the cell size varying in the range $\{1 \dots 8\} \times \{1 \dots 8\}$. This generates roughly 7,000 features per sample. We used the second-order boosting procedure with 200 rounds to train both models (see Section A.3.2). The only difference is in the choice of appropriate loss functions: the exponential $l_1(y, f) = exp(-yf)$ and the sum of exponentials $l_4(y, f) = exp(y - f) + exp(f - y) - 2$ losses (see Table A.1) were used for the classifier and the regressor respectively.

We have chosen the BioID dataset (Jesorsky et al., 2001) as the test dataset because it contains face images captured with a setup close to the one used as the training dataset (BANCA), although significantly more challenging to detect. This dataset contains 1521 images containing only one face in the image taken in different office environments.

The ROC curves are built by varying the threshold τ (see Algorithms 8 and 10) and measuring the detection rate (DR) and the number of false alarms (FA). Multiple detections are integrated

using non-maxima suppression. This is performed iteratively: first we chose the detection with the highest classification score or lowest Jaccard distance and second, we remove the other detections that overlap more than 60% with it.

A.4.2 Results

There are several aspects we investigate: the evolution of the training loss, the face detection performance and the speed of the proposed JScan method. Finally we provide some examples of the proposed detection process produced on the BioID dataset. We would like to point out that the aim of this appendix is to assess the proposed method and not to produce the best possible results. Our goal is to compare a boosted classifier and a boosted regressor on the same datasets. It is clear that improved results can be obtained with more datasets, but it is out of the scope of this study.

Training loss

The training loss evolution provides an insight into how difficult a task is to solve for a particular model. We have plotted in Fig. A.2 the logarithmic evolution of the training loss for the face classifier and the Jaccard distance-based regressor. It can be noticed that the loss decreases exponentially in the case of classification, but only linearly in the case of regression. This suggests that it is significantly harder to learn how far a sub-window is from the ground truth than classifying it as face or background. Still, a slowly increasing accurate regression output is produced as the number of boosting rounds increases.

Face detection performance

We have evaluated the face detection performance of our method JScan and the baseline SScan on the BioID dataset. The sliding-windows approach depends on the search space parameters for location and scale. Hence, we have used two scenarios: the coarse search ($dx = 0.25$, $dy = 0.25$, $ds = 0.20$) and the fine search ($dx = 0.20$, $dy = 0.20$, $ds = 0.10$), which we denote as *SScan (coarse)* and *SScan (fine)* respectively.

The logarithmic ROC curves are plotted in Fig. A.3. It can be noticed that the performance of the SScan method clearly depends on the search parametrization: the fine search significantly

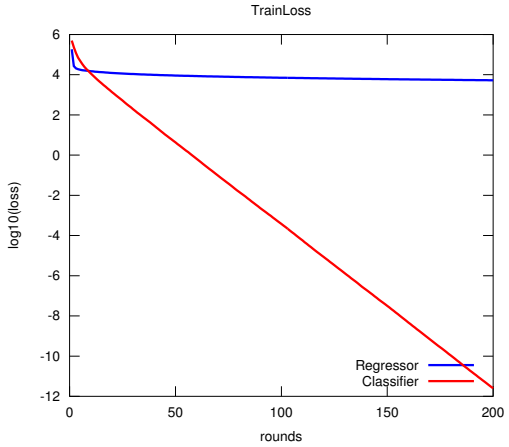


Figure A.2. The logarithmic evolution of the training loss for the face classifier (red) and the Jaccard distance-based regressor (blue).

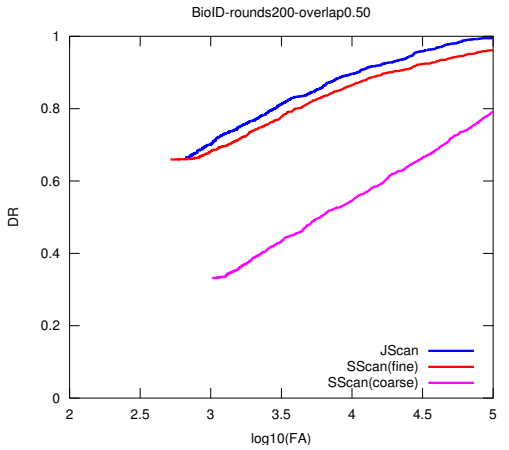
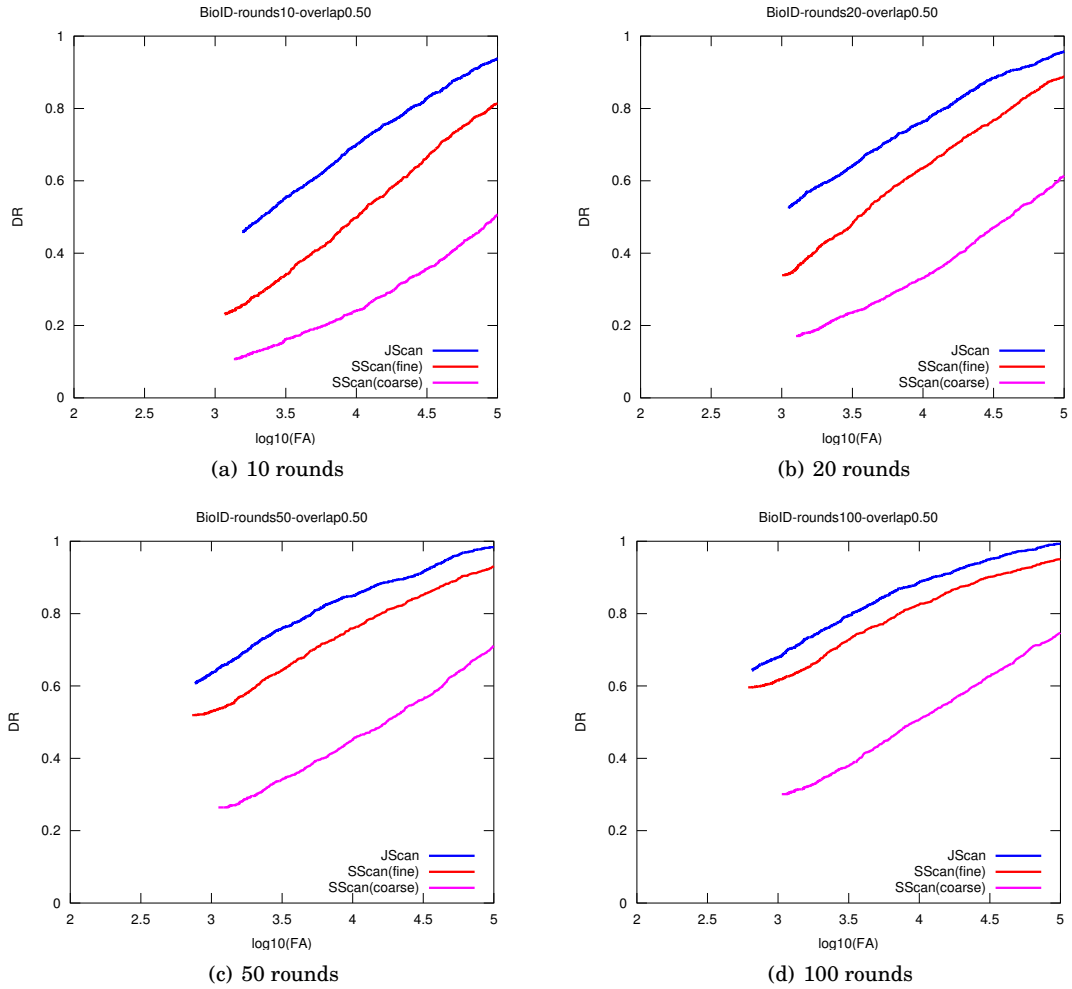


Figure A.3. The logarithmic ROC curves for the BioID dataset using JScan (blue) and SScan with coarse (magenta) and fine (red) search parametrization. All models were trained using 200 boosting rounds.

outperforms the coarse search. This is at the cost of a slower face detector, because the number of sub-windows increases rapidly with the search parameters.

The proposed JScan method performs significantly better than the baseline with a DR which is 5% higher for the same number of false alarms. This performance is maintained for various number of boosting rounds (see Fig. A.4) with a noticeable larger improvement for small number of boosting rounds. This correlates with the evolution of the training loss (see Fig. A.2) when the regressor learns faster for the first rounds, but significantly slower for large number of boosting rounds. This allows the classifier to close the gap in terms of performance.



(a) 10 rounds (b) 20 rounds

(c) 50 rounds (d) 100 rounds

Figure A.4. The logarithmic ROC curves for the BioID dataset using various number of boosting rounds. The results for JScan are plotted with blue, while for SScan with coarse and fine search parametrization with magenta and red, respectively.

Boosting rounds	10	20	50	100	200
			BioID		
JScan	2,941	2,668	2,462	2,388	2,347
SScan (coarse)	8,485	8,485	8,485	8,485	8,485
SScan (fine)	21,055	21,055	21,055	21,055	21,055
Speed-up factor	**2.88**	**3.18**	**3.44**	**3.55**	**3.61**

Table A.2. The number of processed sub-windows (in thousands) for the BioID dataset using various number of boosting rounds. The JScan speed-up factor is computed relative to SScan (coarse).

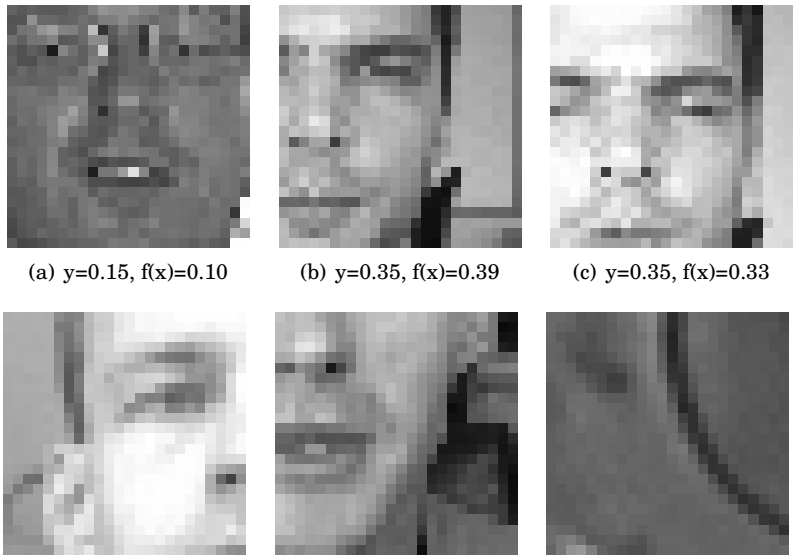
Detection speed

We have also studied the number of processed sub-windows for each method (see Table A.2) with a varying number of boosting rounds. It can be noticed that as the number of boosting rounds increases, the JScan method processes fewer sub-windows for both test datasets. This is because the Jaccard distance-based regressor becomes more reliable and fewer sub-windows need to be refined (see Algorithm 10). This contrasts with both SScan instances that process the same number of sub-windows. Hence, it is possible to achieve an even higher speed with a more accurate regressor. We conclude that our proposed method JScan is significantly faster than the baseline methods for a similar complexity of the model. Indeed, the classifier and the regressor contain the same number of parameters.

Examples

Figure A.5 presents some sub-windows processed by our proposed method on the BioID dataset. The number on the left of the caption (y) is the Jaccard distance and the number on the right $f(x)$ is the estimated one. These samples contain faces at different location displacements and scale variations. This makes the Jaccard distance modelling a more difficult task than face classification. For example the samples b, c, d and e (see Fig. A.5) are excluded when training the face classifier because they are ambiguous. But the Jaccard distance model must cope with these difficult samples. This results in significant errors at testing, but still the predictions are accurate enough for successfully guiding the refinement of potential face detections (see Fig. A.6).

As shown in Fig. A.6, the proposed detection refinement stage concentrates the effort on the most promising locations (hopefully around the ground truth face locations). This is because the number of detections to refine decreases at each step: the ones with a score smaller than 0.50 and



(a) y=0.15, f(x)=0.10 (b) y=0.35, f(x)=0.39 (c) y=0.35, f(x)=0.33

(d) y=0.62, f(x)=0.63 (e) y=0.64, f(x)=0.59 (f) y=0.98, f(x)=0.93

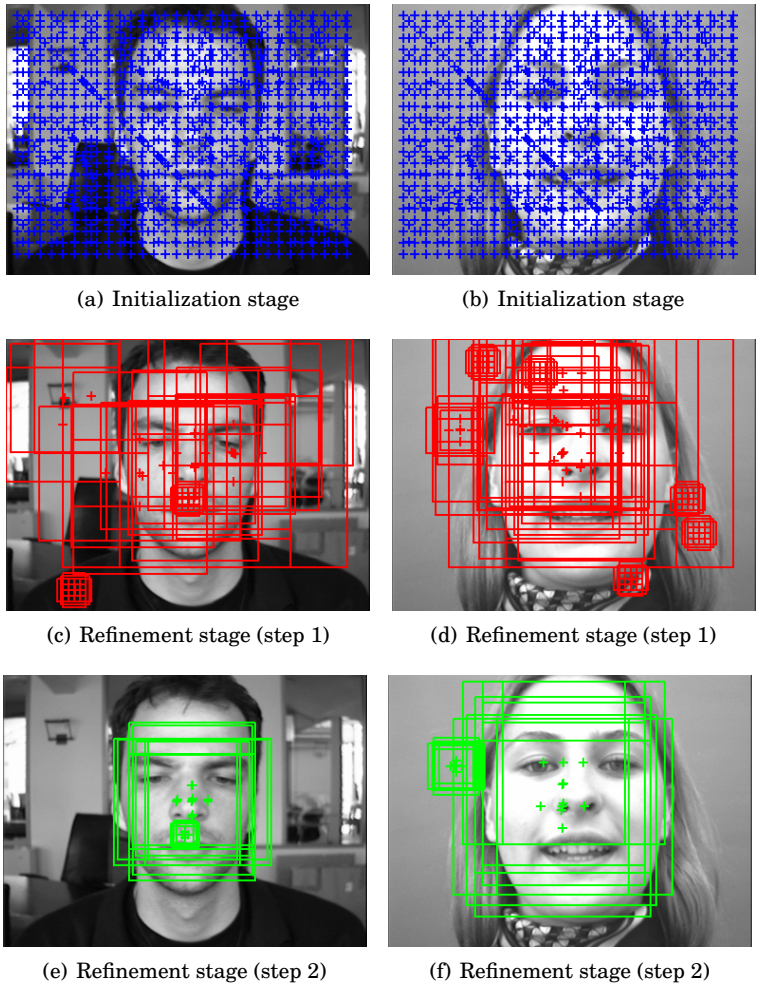
Figure A.5. Examples of sub-windows (x) processed by the JScan method on the BioID dataset. The number on the left of the caption (y) is the Jaccard distance and the number on the right $f(x)$ is the estimated one.

0.40 for the first and second step respectively. This corresponds to detections that are closer, in terms of the Jaccard distance, than 0.50 and 0.40 respectively from the ground truth.

A.5 Conclusions

In this appendix we presented a new face detection method. We trained a model to learn the Jaccard distance between a sub-window and the ground truth location. For this we generalized the boosting algorithm for binary classification to optimize any smooth loss function. Then the binary classifiers and single output regressors were trained with the same algorithm, the only difference being the choice of appropriate loss function.

The experimental results have shown that our face detector processes significantly fewer sub-windows than the baseline sliding-windows approach using an equivalently complex classifier. The face detection performance is improved over the baseline on the BioID dataset with a DR which is 5% higher for the same number of false alarms. These encouraging results show that the idea is feasible. We plan to perform experiments on other more challenging datasets to further assess its performance.



(a) Initialization stage (b) Initialization stage

(c) Refinement stage (step 1) (d) Refinement stage (step 1)

(e) Refinement stage (step 2) (f) Refinement stage (step 2)

Figure A.6. Illustration of the detection process with the JScan method for two images (left and right column respectively). On the first row we display the centres of the initialized detections, while on the second and third rows the refined detections in the first and the second step respectively.

Another interesting finding is that the number of sub-windows to process decreases with the number of boosting rounds. This suggests that a more accurate Jaccard distance regressor would improve the proposed detection method and consequently process fewer sub-windows which would result in a faster face detector. To achieve this we envisage several directions for future work including: boosting more powerful weak learners, faster optimization with respect to the number of boosting rounds and adapting bootstrapping from classification to regression.

Appendix B

A principled approach to remove false alarms by modelling the context of a face detector

In this appendix we present a new method to enhance object detection by removing false alarms in a principled way with few parameters. The method models the output of an object classifier which we consider as the *context*. A hierarchical model is built using the detection distribution around a target sub-window to discriminate between false alarms and true detections. The specific case of face detection is chosen for this work as it is a mature field of research. We report results that are better than baseline methods on XM2VTS and MIT+CMU face databases and significantly reduce the number of false acceptances while keeping the detection rate at approximately the same level.

B.1 Objectives and motivations

A variety of applications like video surveillance, biometric recognition and human-machine interface systems depend on robust face detection algorithms. In the last decade there has been an increasing interest in real-time systems with high accuracy and many successful methods have been proposed (Zhao et al., 2003). Still face detection remains a challenging problem and there are

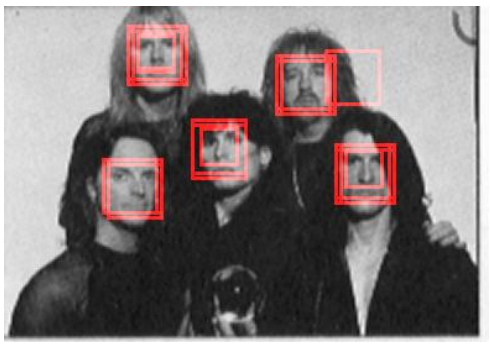


Figure B.1. Typical face detections using the multiscale approach and the boosted cascade classifier described in (Froba and Ernst, 2004) (without clustering multiple detections nor removing false alarms).

improvements to be made.

Face detection is often posed as the task of classifying a sub-window as being a particular object or not. As such it requires a method to sample an image for sub-windows and a classifier to classify the sub-window. Research to date has mainly dealt with the issue of building a robust and accurate object classifier. An object classifier tells if an object is found at a specific position and scale (referred as sub-window) in an image. For instance work by Froba et al. (Froba and Ernst, 2004) and Viola and Jones (Viola and Jones, 2001) has provided significantly improved face classifiers. Different approaches have been proposed such as the pioneering work from Rowley et al. (Rowley et al., 1998) or (Garcia, 2004) based on Neural Networks. But the most successful face detection methods are based on a cascade of boosted classifiers that provide real-time performance with high accuracy (Lienhart and Kuranov, 2003).

There are many ways to obtain sub-windows from an image, with the sliding window approach (Rowley et al., 1998) being the most well known. The sliding window approach finds all the object instances by scanning the image at different positions and scales. This can result in multiple detections and false alarms as shown in Fig. B.1. A merging and pruning heuristic algorithm is then typically used to output the final detections (Rowley et al., 1998; Viola and Jones, 2001; Rodriguez, 2006).

Recent work has been done to overcome the limitations of the sliding window approach by using a branch-and-bound technique to evaluate all possible sub-windows in an efficient way (Lampert et al., 2008). The authors build a model that also predicts the location of the object (Blaschko and Lampert, 2008). However, it is not clear how to use this method for different classifier types (for

instance boosted cascades) or to detect multiple objects.

A different approach was recently proposed in (Takatsuka et al., 2006) and (Takatsuka et al., 2007) where the authors study the score distribution in both location and scale space. Their experimental results have shown that the score distribution is significantly different around a true object location than around a false alarm location, thus making possible to build a model to better distinguish the false alarms and enhance detection. This approach is motivated by the fact that the object classifier is usually trained with geometrically normalized positive samples and it does not process the *context* (area around given samples). Also, some false alarm sub-windows may have a higher score than a true detection nearby and may be selected erroneously as being final detections when using a simple heuristic merging technique.

We propose a model to enhance a given face classifier by discriminating false detections (sub-windows) from true detections using the contextual information. Our approach was inspired from the work of (Takatsuka et al., 2006, 2007). Similarly we investigate the detection distribution around some sub-window (which we call the *context*) in order to evaluate if it corresponds to a true detection or not.

There are significant differences between this work and that presented in (Takatsuka et al., 2006, 2007). The first is that we extract more information from the detection distribution than just the score of the face classifier. For example we count detections within the context and we use features that describe the geometry of the detections around a sub-window. The second significant difference is that we extract features from every possible axis combination (locations x, y and scale s) and we train a classifier to automatically choose the most discriminant features.

B.2 Context-based modelling for face detection

In this section we present a model to discriminate false detections from true detections. First we describe how we *sample* around a target sub-window to build its context. Then we present the *features* we extract from the context and finally the *classifier* that uses these features to discriminate false alarms from true detections.

B.2.1 Sampling

We sample in the 3D space of location (x, y) and scale (s) to collect detections *around* a target sub-window $T_{sw} = (x, y, s)$. For this we vary its position and scale in all directions (left, right, up, down, smaller and bigger) and we form new sub-windows. Those sub-windows that pass the object classifier are gathered with the associated classifier output ms, referred to as the model score in this appendix. We obtain a collection of 4D points $C(T_{sw}) = \{(x_i, y_i, s_i, ms_i)_{i=1,..}\}$ that we call the *context of the target sub-window* T_{sw}. Its parameters are the number of points to be considered on each axis (location and scale) along the positive direction, which we define as N_x, N_y and N_s respectively.

We have used two strategies for context sampling: full and axis. The *full* strategy consists of sampling by varying the location and scale at the same time. In this case the context can have at most $N_{full} = (2N_x + 1) \times (2N_y + 1) \times (2N_s + 1)$ points. In the *axis* strategy the sampling is done just along one axis at a time. This reduces the maximum size of the context to $N_{axis} = (2N_x + 1) + (2N_y + 1) + (2N_s + 1)$ points.

In our experiments we have used $N_x = N_y = 6$ and $N_s = 7$ with 5% increments both in scale and position [1]. This makes N_{axis} (at most 41 points) approximately 60 times smaller than N_{full} (at most 2535 points). The axis sampling approach is better suited for real time applications where building the full context may be too expensive. In our experiments this method has a small performance degradation compared to the full sampling method, but it can be many times faster.

B.2.2 Feature vectors

In the next step we extract a fixed number of low dimensional feature vectors from $C(T_{sw})$. The feature vectors are defined by their attribute(s) and the axis (and axes) used to obtain the attribute(s).

We use 5 attributes that capture the *global information* (counts), the *geometry of the detection distribution* (hits) and the *detection confidence* (score) obtained from the face classifier. The *counts* provide a global description of $C(T_{sw})$ by counting detections on some axis combination. The *score (standard deviation and amplitude)* describes the classifier confidence variation across position and scale changes. The *hits (standard deviation and amplitude)* capture the spread of detections on some axis. The last attribute addresses the intuition that detections can be obtained by varying

[1] The context for a detection of size 100x100 pixels is obtained by sampling sub-windows from approximately 70x70 to 140x140 pixels and translated by at most 34 pixels.

more the scale or the position around a true detection than on the false alarms.

Each feature vector is computed on some axis combination (x, y and s) which gives 7 possible combinations. For example we can build sub-windows by varying all axes, just two of them (like keeping the $scale$ constant and varying only the sub-window's x and y coordinates) or just one of them (like keeping the x and $scale$ fixed and moving the sub-window up and down). More details can be found in Section B.3.2, where we have visualized and investigated the discriminative properties of these features.

B.2.3 Classifier

The context features from the previous section are used to train a classifier to distinguish between false alarms and true detections based on their context. We build a linear classifier for each context feature (described in Section B.2.3) and then we combine them to produce the final result (described in Section B.2.3).

Our aim is to *automatically* select the best attributes and axes that are more discriminant. This makes the context-based model independent of the specific geometric properties of the object to detect, the type of the object classifier or the scanning procedure.

Context classifiers

The contextual information is used to form 35 different context features: there are $n = 5$ types (as discussed in Section B.2.2) computed for each of the $m = 7$ axis combinations. For each feature vector we build a logistic linear model which we denote as $M(x, w)$, where the x is the d-dimensional sample feature vector and w is the $d + 1$-dimensional parameter value. The model output is:

$$M\left(x, w\right) = \frac{1}{1 + exp\left(-w_0 + \sum_{i=1}^{d} x_i w_i\right)}, \tag{B.1}$$

where w_0 is sometimes called the bias term and the w_i terms are the weights of the inputs.

Training the model is done by minimizing the negative of the likelihood of the model output being generated from the input data. Additional L_1 and L_2 norm regularization terms are added as

described in (Lee et al., 2006). Following (Perkins et al., 2003), our function to optimize is:

$$E\left(w,\lambda_1,\lambda_2\right)=\frac{\sum l\left(w,x^+\right)}{N^+}+\beta\frac{\sum l\left(w,x^-\right)}{N^-}+\lambda_1\underbrace{\sum_{i=1}^{n}|w_i|}_{L_1}+\lambda_2\underbrace{\sum_{i=1}^{n}|w_i|^2}_{L_2}, \tag{B.2}$$

where $l\left(w,x\right)=-y\,log(M(x,w))-(1-y)\,log(1-M(x,w))$ is the negative log likelihood of the sample x using the model weights w; obviously y relates to the label of interest so it represents the positive class for the case of $l\left(w,x^+\right)$ and the negative class for $l\left(w,x^-\right)$. The log likelihoods are averaged separately over the N^+ positive samples and the N^- negative samples respectively because of the unbalanced nature of the training samples, λ_1 and λ_2 are priors for the L_1 and L_2 norms. The purpose of the L_2 norm regularization term is to avoid over fitting, while the L_1 one is to keep the model sparse hopefully by automatically selecting the most informative features.

The weight β represents the relative importance attributed to the error caused by the negative samples relative to the one caused by the positive samples. In the case of object detection (in particular face detection) it is preferred to have higher false alarms than to miss objects. This implies that β needs to penalize false rejections more than false acceptances which corresponds to $\beta<1$. Several preliminary experiments were performed on a small sub-set of the training data and $\beta=0.3$ was chosen as the optimal value.

There are some robust methods to optimize the non-continuously differentiable function $E\left(w,\lambda_1,\lambda_2\right)$ (for a review see (Lee et al., 2006)). We have used a simple method called *grafting* described in (Perkins et al., 2003). This method integrates well with standard convex optimization algorithms and it uses an incremental approach to feature selection that suits our needs. Jorge Nocedal's libLBFGS library (Nocedal and Jorge, 1989) was used for the optimization of the error function at each step of the grafting algorithm.

Another related problem we need to solve is the choice of the λ_1 and λ_2 prior terms. For this we use a cross-validation technique on two datasets, one for training and one for tuning, as specified by each database's protocol. We first optimize the λ_1 prior term using a logarithmic scale keeping $\lambda_2=0$ and second we optimize the λ_2 prior term using the same logarithmic scale and keeping the already estimated λ_1 value. The criterion to choose the best (λ_1,λ_2) configuration is the Weighted

Error Rate (WER) defined as:

$$WER(\beta, \tau) = \frac{\beta \times FAR + FRR}{\beta + 1}, \tag{B.3}$$

where FAR is the False Acceptance Rate and FRR is the False Rejection Rate computed as $FRR = 1 - TAR$; TAR is the True Acceptance Rate also referred to as Detection Rate (DR). The same weight β was used as in Equation B.2.

Combined classifier

Each feature classifier can be considered as an expert. By combining them two benefits can be obtained: first the combined classifier should perform better and second only some (the best) experts are combined which implies that some irrelevant features can be (automatically) discarded. The combined model uses the same logistic linear model as for the context classifiers. This makes the proposed hierarchical model a non-linear mapping of the inputs, while each context classifier is kept very simple and linear.

The inputs to the combined classifier are the normalized outputs of the context classifiers. Let us define the context classifiers as $M_{k,l}(x, w)$, where k indicates the attribute type $(k = 1..n, n = 5)$ and l corresponds to the axis combination $(l = 1..m, m = 7)$. Let $\tau_{k,l}$ be the optimum threshold value of the $M_{k,l}$ model. Then the value forwarded to the combined classifier is $x_{k,l} = M_{k,l}(x, w) - \tau_{k,l}$.

This normalization has two benefits. First, the sign indicates the decision of the $M_{k,l}$ model: positive for true detections and negative for false alarms. Second, the absolute value is (empirically) proportional to the confidence of the $M_{k,l}$ model in its decision.

B.3 Experiments

The experimental procedure used in this appendix is defined by these aspects: the databases used, the protocol for these databases, the face classifier and the methods for evaluating performance.

B.3.1 Experimental protocol

We have evaluated our method on two scenarios: XM2VTS (Messer et al., 1999) and MIT+CMU (Rowley et al., 1998). For each scenario a distinct training, tuning and testing image collection was provided to train, optimize parameters and evaluate the context-based model.

The XM2VTS database, split using the Lausanne protocol, contains one large centred face in each image taken in a controlled environment. There is an overlap with the identities used for the training, tuning and testing datasets, but different captures were considered.

The second scenario uses the WEB (Garcia, 2004) database for training, the CINEMA (Garcia, 2004) database for tuning and the MIT+CMU database for testing. This scenario is considered as the most difficult because it consists of images with multiple, sometimes very small, degraded faces or without any face, taken in different environments (indoor and outdoor).

B.3.2 Results and discussions

Face classifier

Our method was tested using the MCT-based face classifier (Froba and Ernst, 2004) implemented with the Torch3vision open-source library [1]. We alter the performance of this face classifier by varying the threshold (θ) of the last stage. This allows us to understand if the performance of the classifier affects the performance of the context models. For each θ four context-based models have been trained: using both *full* and *axis* context sampling methods for each of the two scenarios.

The detections (and contexts) are obtained using a standard sliding-window approach. The context-based model checks each detection and the false alarms are removed. The final detections are obtained by averaging the remaining detections that overlap, which removes most of multiple detections around the same face. It should be noted that no sub-window heuristic pruning was used during scanning.

We have compared our method with the merging method implemented by Torch3vision and referred to as HMergeT. More details can be found in (Rodriguez, 2006). To label a detection as positive we used the Jesorsky measure with the threshold $\epsilon_J = 0.25$ (Jesorsky et al., 2001).

[1] The Torch3vision library is freely available at `http://torch3vision.idiap.ch/`

Analysis of context features

Preliminary experiments have been carried out to analyse if the proposed features provide enough information to discriminate between the two cases of contexts. We have plotted some of these context features in Fig. B.2 on the XM2VTS training dataset. To assign a detection (and its context) to the positive or negative class we have used the Jesorsky measure with a relaxed threshold $\epsilon_J = 0.5$. This is because a valid detection should be kept even if it does not match well the true position, but its context captures a significant part of the ground truth. Still, for face detection performance evaluation a more precise $\epsilon_J = 0.25$ has to be used as specified in the previous section.

We found that there is a significant difference between the negative and the positive contexts. This supports our intuition that around a true detection many more detections are generated than around a false alarm. This implies that just by counting detections good discriminative information is obtained. For example in Fig. B.2 (a, b) more than 95% of the negative contexts have their count attribute less than 95% of the positive ones. Also, fewer detections implies much less score variation for negative samples. It can be noticed that negative contexts are more compact around the center, while the positive are much more spread having the standard deviation much higher for the combination of two axes (see Fig. B.2 (c, d)).

We have found experimentally that it is easier to visually separate the two context classes using the full sampling, for example see Fig. B.2 - (a) versus (b), (c) versus d. This is expected because the full sampling gathers many more detections and it is also verified by the next set of experiments where it outperforms with a small margin the axis sampling variant.

Context-based model evaluation

In this set of the experiments we have evaluated how well the context-based model distinguishes between false alarms and true detections. For this we have computed and plotted the WER as shown in Fig. B.3 for the two scenarios. The full (blue) and axis (green) sampling situations are plotted on the same graphic to easily compare them.

The full sampling context-based model performs better than the axis sampling one for the majority of different threshold values. Still this rather small increase in performance requires much larger contexts (2535 versus 41 samples, see Section B.2.1) which impacts on the speed of the overall

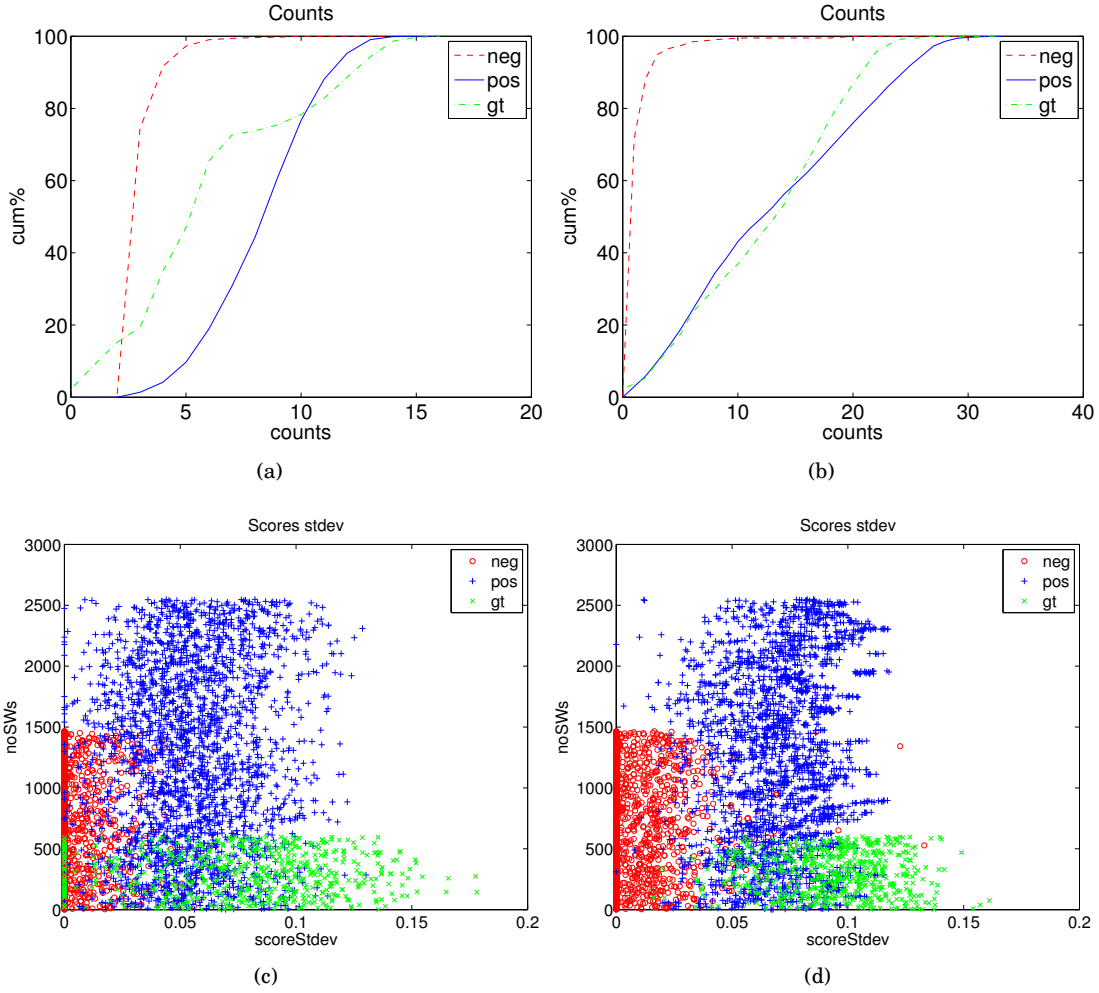
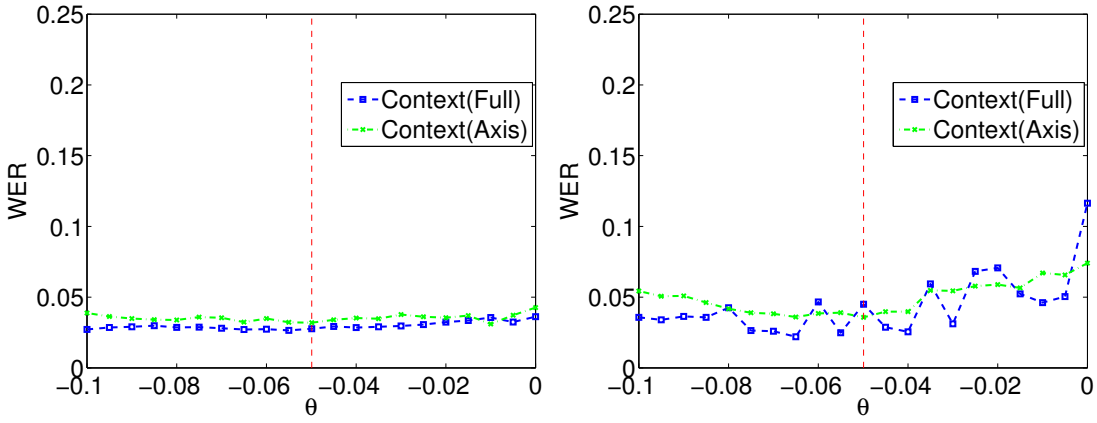


Figure B.2. Distributions of various features using the full (right column) versus axis (left column) sampling on XM2VTS training dataset. Cumulative histogram of counts for two axes (y, scale) using axis sampling (a) and full sampling (b). Cloud of points of score standard deviation for 3 axes (x, y, scale) using axis sampling (c) and full sampling (d). The ground truth is represented with green, the positive class with blue and the negative with red.



(a) XM2VTS-WER (b) MIT+CMU-WER

Figure B.3. Context-based model's weighted error rate (WER) for the test sets of XM2VTS (a) and MIT+CMU (b). The default threshold point of the face classifier is represented with dashed red vertical line.

face detection process. Even with the axis context sampling (41 samples) our context-based model manages to distinguish the false alarms from true detections.

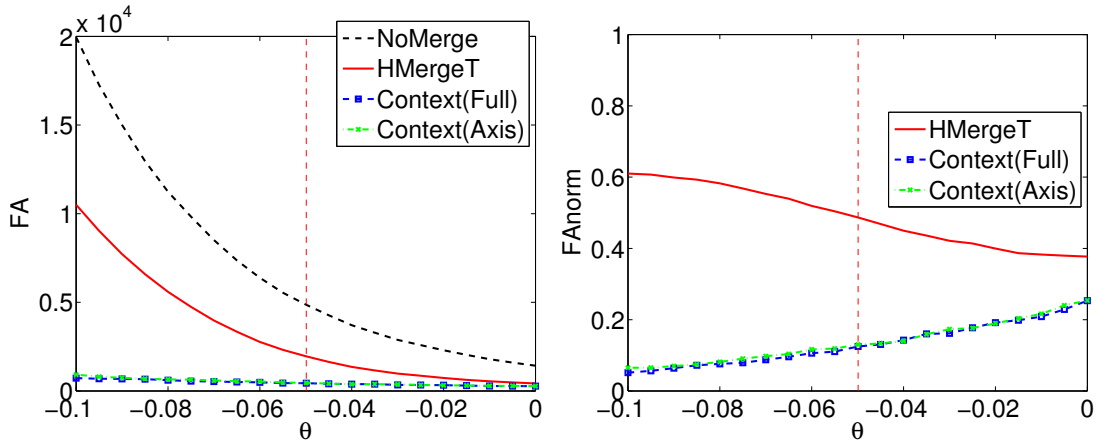
On average both sampling models have an WER lower than 5% for the XM2VTS (Fig. B.3 a) scenario. The same performance is obtained for the MIT+CMU scenario (Fig. B.3 b), even though the training data is scarce (5 and 30 times less training images than for the XM2VTS scenario) and the database is much more challenging.

These results are stable across multiple threshold values of the face classifier. It is important to note that using simple logistic regressions as proposed is enough to obtain an accurate context-based model. This indicates that the features extracted from the contexts (see Section B.2.2), although very simple and low dimensional, are discriminative enough.

Face detection evaluation

Next we have performed experiments to assess the impact our model has on the face detection results. We studied the effect of: i) using the heuristic method HMergeT and ii) using the context-based model with either full or axis sampling, for face detection.

For this we analysed the TAR and the number of false alarms (FA) both parametrized by the threshold of the face classifier: $TAR = TAR(\theta)$ and $FA = FA(\theta)$ respectively. We omit the threshold of the context-based classifier in this parametrization because it is automatically optimized on the tuning dataset (see Section B.2.3) and it is not varied during experiments. In our case the



(a) XM2VTS-FA (b) XM2VTS-FAnorm

Figure B.4. The normalized FA (b) on the XM2VTS scenario. The default threshold point is represented with dashed red vertical line.

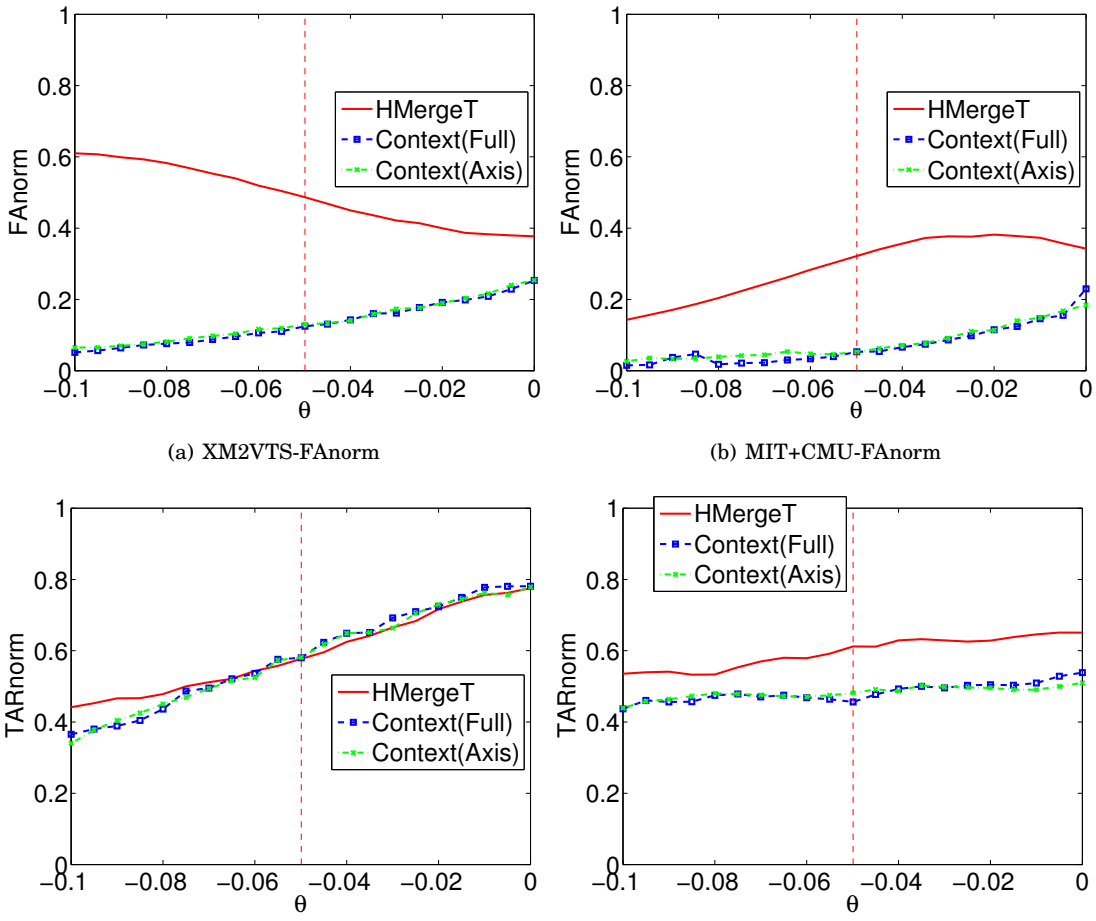
threshold to vary is θ, which is **not** the discriminative threshold. Indeed it just provides different context distributions to train our context-based model. We have chosen these two criteria (TAR and FA) instead of ROC curves, because the significant decrease in FA (see Fig. B.4) makes the ROC curve very skewed to the left.

The aim of any multiple detection clustering algorithm is to remove as few as possible true detections and remove as many as possible false alarms. This motivates the comparison of our approach and the baseline with the face classifier without any merging. Let use define the TAR and the FA of the face classifier without any merging (NoMerge as in Fig. B.4) as TAR_n and FA_n respectively. Then we report the normalized TAR and FA as:

$$FAnorm\,(\theta) = \log\left(1 + \frac{FA\,(\theta)}{FA_n\,(\theta)}\right), \tag{B.4}$$

$$TARnorm\,(\theta) = \frac{TAR\,(\theta)}{TAR_n\,(\theta)}. \tag{B.5}$$

First we analysed the logarithmically normalized FA plots presented in Fig. B.5 (a & b) for the two scenarios. As expected the number of FAs is greatly reduced, with at least an order of magnitude compared to the baseline HMergeT. The significant decrease in the number of FAs demonstrates that our proposed method successfully discriminates false alarms from true detections. An-



(a) XM2VTS-FAnorm

(b) MIT+CMU-FAnorm

(c) XM2VTS-TARnorm

(d) MIT+CMU-TARnorm

Figure B.5. Normalized FA (top row) and normalized TAR (bottom row) plots on XM2VTS (a, c) and MIT+CMU (b, d) scenarios.

other important observation is that there is no significant difference between the full sampling and the axis sampling methods in the number of FAs. This indicates that a context with fewer samples (thus faster to evaluate) can be designed to have similar results.

Second we evaluated the impact on the normalized TAR as presented in Fig. B.5 (c & d). The TAR decreases slightly for the XM2VTS scenario (up to 5%) and is more accentuated for the MIT+CMU scenario (up to 10%).

We conclude that overall our system performs well compared to the baseline, the drop in TAR being justified by the exponential decrease in the FAs. Overall we found no significant performance difference between the full and the axis sampling methods.

B.4 Conclusions

This work has presented a new method to enhance object detection by removing false alarms in a principled way with few parameters. We have evaluated the performance of our method on several popular face databases using a well known face detector to study the effect of two sampling methods - full and axis. It was found that our system reduces the FA exponentially while keeping the TAR at similar level as the baseline approach. The full sampling method has a slightly better performance but it needs many more samples, while the axis sampling version is a trade-off between performance and speed.

There are several advantages to using our method. First our algorithm can be initialized with any sub-window collection, which can be obtained using some sliding window approach or a totally different approach. Second it can work on top of any object classifier - there are no restrictions regarding its score values, its type or the features used. Further improvements can be envisaged including the use of higher dimensional context-based features, different feature classifiers (such as SVM and AdaBoost) or more efficient sampling methods. Other work could also examine the use of contextual information to improve the accuracy of detections and even to recover mis-aligned detections.

Bibliography

E. Bailly-Bailliére, Samy Bengio, F. Bimbot, M. Hamouz, Josef Kittler, J. Mariéthoz, Jiri Matas, Kieron Messer, V. Popovici, F. Porée, and Others. The BANCA database and evaluation protocol. In *Audio-and Video-Based Biometric Person Authentication*, pages 1057–1057. Springer, 2003.

Matthew Blaschko and Christoph Lampert. Learning to Localize Objects with Structured Output Regression. *Learning*, 5302:2–15, 2008.

D Cristinacce and T Cootes. Boosted regression active shape models. *Proceedings of the British Machine Vision Conference*, 2:880–889, 2007.

D Cristinacce and T F Cootes. Facial Feature Detection and Tracking with Automatic Template Selection. *7th International Conference on Automatic Face and Gesture Recognition FGR06*, pages 429–434, 2006.

David Cristinacce and Tim Cootes. Facial feature detection using AdaBoost with shape constraints. *Biomedical Engineering*, 1:231–240, 2003.

David Cristinacce and Tim Cootes. Automatic feature localisation with constrained local models. *Pattern Recognition*, 41(10):3054–3067, 2008. ISSN 00313203.

Mark Culp, George Michailidis, and Kjell Johnson. On Adaptive Regularization Methods in Boosting. *Journal of Computational and Graphical Statistics*, pages 1–22, 2010.

Piotr Doll and Welinder Pietro. Cascaded Pose Regression. *Evaluation*, pages 1078–1085, 2010.

N. Duffy and D. Helmbold. Boosting methods for regression. *Machine Learning*, 47(2):153–200, 2002.

Mark Everingham and Andrew Zisserman. Regression and classification approaches to eye localization in face images. *7th International Conference on Automatic Face and Gesture Recognition FGR06*, pages:441–448, 2006.

Hongliang Fei and Jun Huan. Boosting with Structure Information in the Functional Space : an Application to Graph Classification. In *Optimization*, pages 643–651, 2010.

R. Fergus and P. Perona. Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories. *2004 Conference on Computer Vision and Pattern Recognition Workshop*, 106(1):178–178, 2007.

Y Freund. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. In *Journal of Computer and System Sciences*, volume 55, pages 23–37. Springer, August 1995.

J H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001. ISSN 00905364.

Jerome Friedman, Robert Tibshirani, and Trevor Hastie. Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors). *The Annals of Statistics*, 28(2): 337–407, April 2000. ISSN 0090-5364.

B Froba and Andreas Ernst. Face detection with the modified census transform. *Sixth IEEE International Conference on Automatic Face and Gesture Recognition 2004 Proceedings*, pages 91–96, 2004.

Christophe Garcia. Convolutional face finder: A neural architecture for fast and robust face detection. *Pattern Analysis and Machine*, 26(11):1408–23, November 2004. ISSN 0162-8828.

Ralph Gross, Iain Matthews, Jeff Cohn, Takeo Kanade, and Simon Baker. Multi-PIE. *Proceedings of the International Conference on Automatic Face and Gesture Recognition*, 28(5):807–813, May 2010. ISSN 1541-5058.

Di Huang, Caifeng Shan, Mohsen Ardabilian, Yunhong Wang, and Liming Chen. Local binary patterns and its application to facial image analysis: A survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 41(6):765–781, 2011.

Paul Jaccard. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

Oliver Jesorsky, Klaus J Kirchberg, and Robert W Frischholz. Robust Face Detection Using the Hausdorff Distance. *Computer*, 2091(June):90–95, 2001.

Jean Keomany. Active Shape Models Using Local Binary Patterns. 2006. URL `http://publications.idiap.ch/index.php/publications/show/246`.

Christoph Lampert, Matthew Blaschko, and Thomas Hofmann. Beyond Sliding Windows: Object Localization by Efficient Subwindow Search. *IEEE Conference on Computer Vision and Pattern Recognition (2008)*, 12346(2):1–8, 2008.

S.I. Lee, Honglak Lee, Pieter Abbeel, and A.Y. Ng. Efficient L1 Regularized Logistic Regression. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 401. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

Stan Z. Li, ZhenQiu Zhang, Heung-Yeung Shum, and HongJiang Zhang. Floatboost learning for classification. In *Advances in Neural Information Processing Systems*, pages 993–1000, 2002.

Shengcai Liao, Xiangxin Zhu, Zhen Lei, Lun Zhang, and Stan Z Li. Learning Multi-scale Block Local Binary Patterns for Face Recognition. *Learning*, 4642:828–837, 2007.

Rainer Lienhart and Alexander Kuranov. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. *Pattern Recognition*, 2003.

Hamed Masnadi-Shirazi. On the design of robust classifiers for computer vision. *IEEE Conference on Computer Vision and Pattern Recognition (2008)*, pages 779–786, 2010. ISSN 10636919.

L Mason, J Baxter, PL Bartlett, and M. Frean. Functional gradient techniques for combining hypotheses. *Advances in Neural Information Processing Systems*, pages 221–246, 1999a.

L Mason, J Baxter, P Bartlett, and M Frean. Boosting Algorithms as Gradient Descent. In *Convergence*, volume 12, pages 512–518. MIT Press, 2000.

Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent in function space. In *Neural information processing systems*, volume 12, pages 512–518. Research School of Information, ANU Canberra, Citeseer, 1999b.

K Messer, J Matas, J Kittler, J Luettin, and G Maitre. XM2VTSDB: The extended M2VTS database. In *Second International Conference on Audio and Videobased Biometric Person Authentication*, volume 964, pages 72–77. Citeseer, 1999.

Dong C. Liu Nocedal and Jorge. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 1989.

T Ojala and M Pietikainen. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *Pattern Analysis and Machine Intelligence*, 24(7):971–987, 2002.

Simon Perkins, Kevin Lacker, and James Theiler. Grafting: Fast, Incremental Feature Selection by Gradient Descent in Function Space. *Journal of Machine Learning Research*, 3(7-8):1333–1356, 2003. ISSN 15324435.

Yann Rodriguez. Face detection and verification using local binary patterns. *Ecole Polytechnique Federale de Lausanne, PhD*, 3681, 2006.

H A Rowley, S Baluja, and T Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998. ISSN 01628828.

Mohammad J Saberian. Boosting Classifier Cascades. *Advances in Neural Information Processing Systems 23*, pages 1–9, 2010.

Robert E Schapire. The Boosting Approach to Machine Learning An Overview. *MSRI Workshop on Nonlinear Estimation and Classification*, 7(4):149–172, 2002. ISSN 09300325.

Shai Shalev-Shwartz, Yonatan Wexler, and Amnon Shashua. ShareBoost: Efficient Multiclass Learning with Feature Sharing. *Neural Information Processing Systems*, 2011.

Pannagadatta K Shivaswamy and Tony Jebara. Empirical Bernstein Boosting. *Proocedings of the 13th International Conference on Artificial Intelligence and Statistics AISTATS*, 9:733–740, 2010.

PK Shivaswamy and T. Jebara. Variance Penalizing AdaBoost. *cs.cornell.edu*, pages 1–9, 2011.

Terence Sim and Simon Baker. The CMU Pose, Illumination, and Expression (PIE) Database. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:1615–1618, 2003.

Jan Sochman and Jiri Matas. Waldboost " learning for time constrained sequential detection. In *Computer Vision and Pattern Recognition*, pages 150–156, 2005.

Hiromasa Takatsuka, Masayuki Tanaka, and Masatoshi Okutomi. Spatial merging for face detection. In *SICE-ICASE, 2006. International Joint Conference*, pages 5587–5592. IEEE, 2006. ISBN 8995003855.

Hiromasa Takatsuka, Masayuki Tanaka, and Masatoshi Okutomi. Distribution-based face detection using calibrated boosted cascade classifier. In *14th International Conference on Image Analysis and Processing (ICIAP)*, pages 351–356. IEEE, 2007. ISBN 0769528775.

Xiaoyang Tan and Bill Triggs. Enhanced local texture feature sets for face recognition under difficult lighting conditions. *IEEE transactions on image processing*, 19(6):1635–50, June 2010. ISSN 1941-0042.

A Torralba, K P Murphy, and W T Freeman. Sharing visual features for multiclass and multiview object detection. *The IEEE Conf on Computer Vision and Pattern Recognition CVPR*, 29(5):854–869, 2007. ISSN 01628828.

Jiri Trefny and Jiri Matas. Extended Set of Local Binary Patterns for Rapid Object Detection. *Computer Vision Winter Workshop*, 1:1–7, 2010.

Michel Valstar and Xavier Binefa. Facial Point Detection using Boosted Regression and Graph Models. *Computing*, pages 2729–2736, 2010. ISSN 10636919.

P Viola and M Jones. Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR 2001*, 1(C):I–511–I–518, 2001. ISSN 10636919.

P Viola and M Jones. Fast and Robust Classification using Asymmetric AdaBoost and a Detector Cascade. *Advances in Neural Information Processing Systems*, 2:1311–1318, 2002. ISSN 10495258.

Paul Viola and M.J. Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

Markus Weber. Assorted scenes around the Caltech campus and in the Vision lab. URL `http://www.vision.caltech.edu/html-files/archive.html`.

Z J Xiang, Y T Xi, U Hasson, and P J Ramadge. Boosting with Spatial Regularization. *Advances in Neural Information Processing Systems*, pages 1–9, 2009.

Lun Zhang, Rufeng Chu, Shiming Xiang, Shengcai Liao, and S Li. Face Detection Based on Multi-Block LBP Representation. *Advances in Biometrics*, 4642:11–18, 2007.

W Zhao, R Chellappa, and PJ Phillips. Face recognition: A literature survey. *ACM Computing Surveys*, 35(4):399–458, 2003.

Curriculum Vitae

Name: Cosmin Atanasoaei

Age: 29

Nationality: Romanian

Permanent Address: Rue de la Paix, 6, Yverdon-les-Bains 1400, Switzerland.

Email: accosmin@gmail.com

General area of interest and expertise:

Machine learning, boosting, object detection, image processing

C++ software design, implementation efficiency

Education:

1. *May 2008 - April 2012*

   PhD in Electrical Engineering at the Ecole Polytechnique Fédérale de Lausanne, Switzerland and the Idiap Research Institute, Martigny under the supervision of Prof. Hervé Bourlard, Dr. Sébastien Marcel and Dr. Christopher McCool (defense scheduled: 27th April 2012).

2. *October 2006 - February 2008*

   Master of Computer Science at the Politechnica University, Bucharest, Romania.

3. *October 2001 - July 2006*

   Bachelor of Computer Science at the Politechnica University, Bucharest, Romania.

Professional Experience:

1. *May 2008 - April 2012*

   Research Assistant at the Idiap Research Institute, Martigny, Switzerland.

2. *May 2007 - May 2008*

   C++ Software Developer at AVIRA Soft SRL, Bucharest, Romania.

3. *October 2006 - May 2007*

   C++ Software Developer at ORSYP Software, Bucharest, Romania.

4. *July 2003 - October 2006*

   C++ Software Developer at DIGITAIR SRL, Bucharest, Romania.

Research Projects directly involved in:

1. MOBIO MObile BIOmetry (European FP7 project), www.mobioproject.org
2. Context Modelling for Object Detection(CONTEXT), funded by the Hasler Foundation
3. Interactive Multimodal Information Management (IM2), http://www.im2.ch/

Additional expertise:

1. Programming languages: C++, bash, SQL, Java.
2. Technologies: Qt, STL, Boost, MFC.
3. Applications: QtCreator, Code::Blocks, Visual Studio.
4. Operating Systems: UNIX/Linux, Windows.

References:

1. Prof. Hervé Bourlard, Idiap Research Institute, bourlard@idiap.ch
2. Dr. Sebastien Marcel, Idiap Research Institute, Sebastien.Marcel@idiap.ch
3. Dr. Christopher McCool, Idiap Research Institute, christopher.mccool@idiap.ch@idiap.ch

---

List of Publications:

Theses:

1. “Multivariate Boosting with Look-up Tables for Face Processing”, PhD Thesis, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2012.

Conference papers (peer-reviewed):

1. Cosmin Atanasoaei, Christopher McCool and Sébastien Marcel, “A principled approach to remove false alarms by modelling the context of a face detector”, in proceedings of the British Machine Vision Conference (BMVC), 2010

2. Sébastien Marcel, Christopher McCool, Cosmin Atanasoaei, Flavio Tarsetti, J. Pesán, P. Matejka, J. Cernocky, M. Helistekangas, and M. Turtinen, “MOBIO: Mobile biometric face and speaker authentication”, in proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010

3. J. Parris, M. Wilber, B. Helfin, H. Rara, A. El-barkouky, A. Farag, J. Movellan, M. Santana, J. Lorenzo, M.N. Teli, S. Marcel, C. Atanasoaei, and T. Boult, “Face and eye detection on hard datasets”, in the IEEE IAPR International Joint Conference on Biometrics (IJCB), 2011

Technical Reports different from the above:

1. Cosmin Atanasoaei, Christopher McCool and Sébastien Marcel, “On Improving Face Detection Performance by Modelling Contextual Information”, Idiap Research Report Idiap-RR-43-2010, Idiap Research Institute, December 2010

2. Cosmin Atanasoaei, Christopher McCool and Sébastien Marcel, “Face detection using boosted Jaccard distance-based regression”, Idiap Research Report Idiap-RR-02-2012, Idiap Research Institute, January 2012