Journal of Machine Learning Research 15 (2014) 1-41 Submitted 3/12; Revised 10/13; Published 7/14

Boosting Algorithms for Detector Cascade Learning

Mohammad Saberian SABERIAN@UCSD.EDU
Nuno Vasconcelos NUNO@UCSD.EDU
Statistical Visual Computing Laboratory,
University of California, San Diego
La Jolla, CA 92039, USA

Editor: Yoram Singer

Abstract

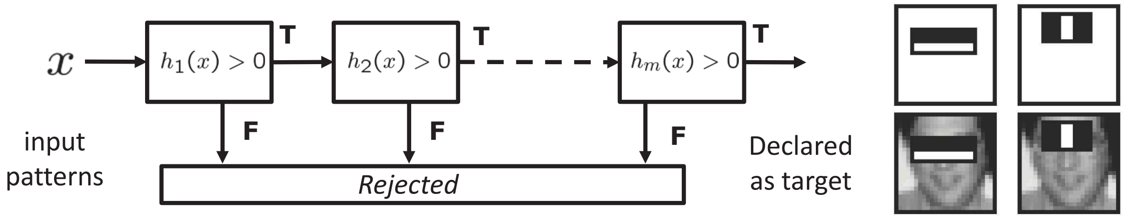
The problem of learning classifier cascades is considered. A new cascade boosting algorithm, *fast cascade boosting* (FCBoost), is proposed. FCBoost is shown to have a number of interesting properties, namely that it 1) minimizes a Lagrangian risk that jointly accounts for classification accuracy and speed, 2) generalizes adaboost, 3) can be made cost-sensitive to support the design of high detection rate cascades, and 4) is compatible with many predictor structures suitable for sequential decision making. It is shown that a rich family of such structures can be derived recursively from cascade predictors of two stages, denoted *cascade generators.* Generators are then proposed for two new cascade families, *last-stage* and *multiplicative* cascades, that generalize the two most popular cascade architectures in the literature. The concept of *neutral predictors* is finally introduced, enabling FCBoost to automatically determine the cascade configuration, i.e., number of stages and number of weak learners per stage, for the learned cascades. Experiments on face and pedestrian detection show that the resulting cascades outperform current state-of-the-art methods in both detection accuracy and speed.

Keywords: complexity-constrained learning, detector cascades, sequential decision-making, boosting, ensemble methods, cost-sensitive learning, real-time object detection

1. Introduction

There are many applications where a classifier must be designed under computational constraints. A prime example is object detection, in computer vision, where a classifier must process hundreds of thousands of sub-windows per image, extracted from all possible image locations and scales, at a rate of several images per second. One possibility to deal with this problem is to adopt sophisticated search strategies, such as branch-and-bound or divide-and-conquer, to reduce the number of sub-windows to classify (Lampert et al., 2009; Vijayanarasimhan and Grauman, 2011; Lampert, 2010). While these methods are compatible with popular classification architectures, e.g., the combination of a support vector machine (SVM) and the bag-of-words image representation, they do not speed up the classifier itself. An alternative solution is to examine all sub-windows but adapt the complexity of the classifier to the difficulty of their classification. This strategy has been the focus of substantial attention since the introduction of the detector cascade architecture (Viola and Jones, 2001). As illustrated in Figure 1 a) this architecture is implemented as a sequence of binary classifiers $h_1(x), \dots h_m(x)$, known as the *cascade stages*. These stages

©2014 Mohammad Saberian and Nuno Vasconcelos.



(a) (b)

Figure 1: (a) detector cascade and (b) examples of weak learners used for face detection (Viola and Jones, 2001).

have increasing complexity, ranging from a few machine operations for $h_1(x)$ to extensive computation for $h_m(x)$. An example x is declared a target by the cascade if and only if it is declared a target by all its stages. Since the overwhelming majority of sub-windows in an image do not contain the target object, a very large portion of the image is usually rejected by the early cascade stages. This makes the average detection complexity quite low. However, because the later stages can be arbitrarily complex, the cascade can have very good classification accuracy. This was convincingly demonstrated by using the cascade architecture to design the first real-time face detector with state-of-the-art classification accuracy (Viola and Jones, 2001). This detector has since found remarkable practical success, and is today popular in applications of face detection involving low-complexity processors, such as digital cameras or cell phones.

In the method of Viola and Jones (2001), cascade stages are designed sequentially, by simply training each detector on the examples rejected by its predecessors. Each stage is designed by boosting decision stumps that operate on a space of Haar wavelet features, such as those shown in Figure 1-b). Hence, each stage is a linear combination of weak learners, each consisting of a Haar wavelet and a threshold. This has two appealing properties. First, because it is possible to evaluate each Haar wavelet with a few machine operations, cascade stages can be very efficient. Second, it is possible to control the complexity of each stage by controlling its number of weak learners. However, while fast and accurate, this detector is not optimal under any sensible definition of cascade optimality. For example, it does not address the problems of 1) how to automatically determine the optimal cascade configuration, e.g., the numbers of cascade stages and weak learners per stage, 2) how to design individual stages so as to guarantee optimality of the cascade as a whole, or 3) how to factor detection speed as an explicit variable of the optimization process. These limitations have motivated many enhancements to the various components of cascade design, including 1) new features (Lienhart and Maydt, 2002; Dalal and Triggs, 2005; Pham et al., 2010; Dollár et al., 2009), 2) faster feature selection procedures (Wu et al., 2008; Pham and Cham, 2007), 3) post-processing procedures to optimize cascade performance (Lienhart and Maydt, 2002; Luo, 2005; Sun et al., 2004), 4) extensions of adaboost for improved design of the cascade stages (Viola and Jones, 2002; Masnadi-Shirazi and Vasconcelos, 2007; Sochman and Matas, 2005; Schneiderman, 2004; Li and Zhang, 2004; Tuzel et al., 2008), 5) alternative cascade structures (Xiao et al., 2003; Bourdev and Brandt, 2005; Xiao et al., 2007; Sochman and Matas, 2005), and 6) joint, rather than sequential stage design (Dundar and Bi, 2007; Lefakis and Fleuret, 2010; Sochman and Matas, 2005; Bourdev and Brandt, 2005). While these advances improved the performance, the optimal design of a whole cascade is still an open problem. Most existing solutions rely on assumptions, such as the independence of cascade stages, that do not hold in practice.

In this work, we address the problem of automatically learning both the configuration and the stages of a high detection rate detector cascade, under a definition of optimality that accounts for both classification accuracy and speed. This is accomplished with the *fast cascade boosting* (FCBoost) algorithm, an extension of adaboost derived from a Lagrangian risk that trades-off detection performance and speed. FCBoost optimizes this risk with respect to a predictor that complies with the sequential decision making structure of the cascade architecture. These predictors are called *cascade predictors*, and it is shown that a rich family of such predictors can be derived recursively from a set of *cascade generator* functions, which are cascade predictors of two stages. Boosting algorithms are derived for two elements of this family, *last-stage* and *multiplicative* cascades. These are shown to generalize the cascades of embedded (Xiao et al., 2003; Bourdev and Brandt, 2005; Xiao et al., 2007; Sochman and Matas, 2005; Masnadi-Shirazi and Vasconcelos, 2007; Pham et al., 2008) or independent (Viola and Jones, 2001; Schneiderman, 2004; Brubaker et al., 2008; Wu et al., 2008; Shen et al., 2011, 2010) stages commonly used in the literature. The search for the cascade configuration is naturally integrated in FCBoost by the introduction of *neutral predictors*. This allows FCBoost to automatically determine 1) number of cascade stages and 2) number of weak learners per stage, by simple minimization of the Lagrangian risk. The procedure is compatible with existing cost-sensitive extensions of boosting (Viola and Jones, 2002; Masnadi-Shirazi and Vasconcelos, 2007; Pham et al., 2008; Masnadi-Shirazi and Vasconcelos, 2010) that guarantee cascades of high detection rate, and generalizes adaboost in a number of interesting ways. A detailed experimental evaluation on face and pedestrian detection shows that the resulting cascades outperform current state-of-the-art methods in both detection accuracy and speed.

The paper is organized as follows. Section 2 reviews the challenges of cascade learning and previously proposed solutions. Section 3 briefly reviews adaboost, the most popular stage learning algorithm, and proposes its generalization for the learning of detector cascades. Section 4 studies the structure of cascade predictors, introducing the concept of cascade generators. Two generators are then proposed, from which two cascade families (last-stage and multiplicative) are derived. The search for the cascade configuration is then studied in Section 5. In this section the Lagrangian extension of the cascade boosting algorithm is introduced, so as to account for detector complexity in the cascade optimization, and a procedure for the automatic addition of cascade stages during boosting is developed, using neutral predictors. All these contributions are consolidated into the FCBoost algorithm in Section 6, whose specialization to last-stage and multiplicative cascades is shown to generalize the two main previous approaches to cascade design. A number of interesting properties of the algorithm are also discussed, and a cost-sensitive extension is derived. Finally, an experimental evaluation is presented in Section 7 and some conclusions drawn in Section 8. An early version of this work was presented in NIPS (Saberian and Vasconcelos, 2010).

2. Prior Work

A large literature on detector cascade learning has emerged over the past decade. In this section, we briefly review the main problems in this area and their current solutions.

2.1 The Problems of Cascade Learning

As illustrated in Figure 1, a cascaded detector is a sequence of detector *stages*. The aim is to detect instances from a *target* class. Examples from this class are denoted *positives* while all others are denoted *negatives*. An example rejected, i.e., declared a negative, by any stage is rejected by the cascade. Examples classified as positives are propagated to subsequent stages. To be computationally efficient, the cascade must use simple classifiers in the early stages and complex ones later on. Under the procedure proposed by Viola and Jones (2001), the cascade designer must first select a number of stages and the target detection/false-positive rate for each stage. A high detection rate is critical, since improperly rejected positives cannot be recovered. The false-positive rate is less critical, since the cascade false-positive rate can be decreased by addition of stages, although at the price of extra computation. The stages are designed with adaboost. The target detection rate is met by manipulating the stage threshold, and the target false-positive rate by increasing the number of weak learners. This frequently leads to an exceedingly complex learning procedure. One difficulty is that the optimal cascade configuration (number of stages and stage target rates) is unknown. We refer to this as the *cascade configuration problem*. While some configurations have evolved by default, e.g., 20 stages, with a detection rate of 99.5% and a false-positive rate of 50%, there is nothing special about these values. This problem is compounded by the fact that, for late stages where negative examples are close to the classification boundary, it may be impossible to meet the target rates. In this case, the designer must backtrack (redesign some of the previous stages). Frequently, various iterations of parameter tuning are needed to reach a satisfactory cascade. Since each iteration requires boosting over a large set of examples and features, the process can be tedious and time consuming. We refer to this as the *design complexity* problem.

Even when a cascade is successfully designed, the process has no guarantees of optimal classification performance. One problem is that, while computationally efficient, the Haar wavelet features lacks discriminant power for many applications. This is the *feature design* problem. This problem is frequently compounded by lack of convergence of adaboost. Note that while adaboost is consistent (Bartlett and Traskin, 2007), there are no guarantees that a classifier with small number of boosting iterations, e.g., early stages of a cascade, will produce classifiers that generalize well. We refer to this as the *convergence problem*. This problem is magnified by the mismatch between the adaboost risk, which penalizes misses/false-positives equally, and the asymmetry of the target detection and false-positive rates used in practice. Although a stage can always meet the target detection rate by threshold manipulation, the resulting false-positive rate can be strongly sub-optimal (Masnadi-Shirazi and Vasconcelos, 2010). In general, better performance is obtained with *asymmetric* learning algorithms, that optimize the detector explicitly for the target detection rate. This is the *cost-sensitive* learning problem. Besides classification optimality, the learned cascade is rarely the fastest possible. This is not surprising, since speed is not an explicit variable of the cascade optimization process. While the specification of stage false-positive rates can be used to shuffle computation between stages, there is no way to predict the amount of computation corresponding to a particular rate. This is the *complexity optimization* problem.

2.2 Previous Solutions

Over the last ten years, significant research has been devoted to all of the above problems.

Feature design: Viola and Jones introduced a very efficient set of Haar wavelets (Viola and Jones, 2001). They showed that these features could be extracted, with a few operations, from an integral image (cumulative image sum). While all features in the original Haar set were axis-aligned, it is possible to extent it for 45° rectangles, (Lienhart and Maydt, 2002). Similarly, several authors pursued extensions to other orientations (Carneiro et al., 2008; Du et al., 2006; Messom and Barczak, 2006). More recently, this has been extended to compute integral images over arbitrary polygonal regions (Pham et al., 2010). Beyond these features, integral images can also be used to efficiently compute histograms (Porikli, 2005). This reduces to quantizing the image into a set of *channels* (associated with the histogram bins) and computing an integral image per channel. For example, a computationally efficient version of the HOG descriptor (Dalal and Triggs, 2005) was then developed and used to design a real-time pedestrian detector cascade (Zhu et al., 2006). More recently, this idea has been extended to multiple other channels (Dollár et al., 2009). Finally, extensions have been developed for more general statistical descriptors, e.g., the covariance features (Tuzel et al., 2008). While the algorithms proposed in this work support any of these features, we adopt the Haar set (Viola and Jones, 2001). This is mostly for consistency with the cascade learning literature, where Haar wavelets are predominant.

Design of stage classifiers: A number of enhancements to the stage learning method of Viola and Jones detector have been proposed specifically to address the problems of convergence rate, cost-sensitive learning, and training complexity. One potential solution to the convergence problem is to adopt recent extensions of adaboost, which converge with smaller numbers of weak learners. Since adaboost is a greedy feature selection algorithm, the effective number of weak learners can be reduced by using forward-backward feature selection procedures (Zhang, 2011) or reweighing weak learners by introduction of sparsity constraints in the optimization (Collins et al., 2002; Duchi and Singer, 2009). This results in more accurate classification with less weak learners, i.e., a faster classifier. While these algorithms have not been used in the cascade learning literature, several authors have used similar ideas to improve stage classifiers. For example, augmenting adaboost with a floating search that eliminates weak learners of small contribution to classifier performance (Li and Zhang, 2004) or by using linear discriminant analysis (LDA) (Shen et al., 2011, 2010). Moreover, by interpreting the boosted classifier as a hyperplane in the space of weak learner outputs, several authors have shown how to refine the hyperplane normal so as to maximize class discrimination. Procedures that recompute the weight of each weak learner have been implemented with SVMs (Xiao et al., 2003), variants of LDA (Wu et al., 2008; Shen et al., 2011, 2010), and non-linear feature transformations (Schneiderman, 2004). The hyperplane refinement usually optimizes classification error directly, rather than the exponential loss of adaboost, further improving the match between learning objective and classification performance. Finally, faster convergence is usually possible with different weak learners, e.g., linear SVMs (Zhu et al., 2006) or decision trees of depth two (Dollár et al., 2009), and boosting algorithms such as realboost or logitboost (Sochman and Matas, 2005; Schneiderman, 2004; Li and Zhang, 2004; Tuzel et al., 2008).

Beyond classification performance, some attention has been devoted to design complexity. Since the bulk of the learning time is spent on weak learner selection, low-complexity methods have been proposed for this. For example, it is possible to trade off memory for computational efficiency (Wu et al., 2008) or to model Haar wavelet responses as Gaussian variables, whose statistics can be computed efficiently (Pham and Cham, 2007). While speeding up the design of each stage, these methods do not eliminate all aspects of threshold tuning, stage backtracking, etc. It could be argued that this is the worst component of design complexity, since these operations require manual supervision. A number of enhancements have been proposed in this area. While Viola and Jones proposed stage-specific threshold adjustments (Viola and Jones, 2001), it is possible to formulate threshold adjustments as an a-posteriori optimization of the whole cascade (Luo, 2005; Sun et al., 2004). These methods are hampered by the limited effectiveness of threshold adjustments when stage detectors have poor ROC performance (Masnadi-Shirazi and Vasconcelos, 2010). Better performance is usually achieved with cost-sensitive extensions of boosting, which optimize a cost-sensitive risk directly (Viola and Jones, 2002; Masnadi-Shirazi and Vasconcelos, 2007; Pham et al., 2008). More recently, Masnadi-Shirazi *et al.* proposed Bayes consistent cost-sensitive extensions of adaboost, logitboost, and realboost (Masnadi-Shirazi and Vasconcelos, 2010). These algorithms were shown to substantially improve the false-positive performance of cascades of high detection rate (Masnadi-Shirazi and Vasconcelos, 2007). These could be combined with the methods which devise a predictor of the optimal false positive and detection rate for each stage, from statistics of the previous stages, so as to design a cascade of cost-sensitive stages automatically (Brubaker et al., 2008; Dundar and Bi, 2007).

Cascade configuration: Most of the above enhancements assume a known cascade configuration and sequential stage learning. This is a suboptimal design strategy and the assumed cascade configuration may not be attainable in practice. An alternative is to adopt cascades of *embedded stages* where each stage is the starting point for the design of the next (Xiao et al., 2003; Bourdev and Brandt, 2005; Xiao et al., 2007; Sochman and Matas, 2005; Masnadi-Shirazi and Vasconcelos, 2007; Pham et al., 2008). The main advantage of this structure is that the whole cascade can be designed with a single boosting run, and adding exit points to a standard classifier ensemble. This also minimizes the convergence rate problems of individual stage design. Using Wald's theory of sequential decision making, it is possible to derive a method for learning embedded stages (Sochman and Matas, 2005). While attempting to optimize the whole cascade, these approaches do not fully address the configuration problem. Some simply add an exit point per weak learner (Masnadi-Shirazi and Vasconcelos, 2007; Xiao et al., 2007; Sochman and Matas, 2005), while others use post-processing (Bourdev and Brandt, 2005; Xiao et al., 2003) or pre-specified detection and false-positive rates (Pham et al., 2008) to determine exit point locations. More recently, it is proposed to learn all stages simultaneously, by modeling a cascade as the product, or logical "AND", of its stages (Lefakis and Fleuret, 2010; Raykar et al., 2010).

Overall, despite substantial progress, no method addresses all problems of cascade learning. Since few approaches explicitly optimize the cascade configuration, fewer among these rely on cost-sensitive learning, and no method optimizes detection speed explicitly, cascade learning can require extensive trial and error. This can be quite expensive from a computational point of view and leads to a tedious design procedure, which can produce sub-optimal cascades. In the following sections we propose an alternative framework, which is fully automated and jointly determines 1) the number of cascade stages, 2) the number of weak learners per stage, and 3) the predictor of each stage, by minimizing a Lagrangian risk that is cost-sensitive and explicitly accounts for detection speed.

3. An Extension of Adaboost for the Design of Classifier Cascades

We start with a brief review of boosting.

3.1 Boosting

A binary classifier $h : \mathcal{X} \rightarrow \{-1, 1\}$ maps an example x into a class label $y(x)$. A learning algorithm seeks the classifier of minimum probability of error, $P_X(h(x) \neq y(x))$, in the space of binary mappings

$$\mathbf{H} = \{h|h : \mathcal{X} \rightarrow \{-1, 1\}\}.$$

Since $\mathbf{H}$ is not convex and $h \in \mathbf{H}$ not necessarily differentiable, this is usually done by restricting the search to mappings of the form

$$h(x) = sign[f(x)],$$

where $f : \mathcal{X} \rightarrow \mathbb{R}$, is a *predictor.* The goal is then to learn the optimal $f(x)$ in a set of predictors

$$\mathbf{F} = \{f|f : \mathcal{X} \rightarrow \mathbb{R}\}.$$

This is the predictor which minimizes the classification risk, $\mathcal{R}_E : \mathbf{F} \rightarrow \mathbb{R}$,

$$\mathcal{R}_E[f] = E_{X,Y}\{L(y(x), f(x))\} \simeq \frac{1}{|S_t|}\sum_i L(y_i, f(x_i)), \tag{1}$$

where $L : \{+1, -1\} \times \mathbb{R} \rightarrow \mathbb{R}$ is a loss function, and $S_t = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ is a set of training examples x_i of labels y_i.

Boosting algorithms are iterative procedures that learn f as a combination of simple predictors, known as *weak learners,* from a set $\mathbf{G} = \{g_1(x), \ldots, g_n(x)\} \subset \mathbf{F}$. The optimal combination is the solution of

$$\begin{cases} \min_{f(x)} & \mathcal{R}_E[f] \\ s.t: & f(x) \in span(\mathbf{G}). \end{cases} \tag{2}$$

Each boosting iteration reweights the training set and adds the weak learner of lowest weighted error rate to the weak learner ensemble. When $\mathbf{G}$ is rich enough, i.e., contains a predictor with better than chance-level weighted error rate for any distribution over training examples, the boosted classifier can be arbitrarily close to the minimum probability of error classifier (Freund and Schapire, 1997). For most problems of practical interest, $\mathbf{G}$ is an overcomplete set and the solution of (2) can have many decompositions in $span(\mathbf{G})$. In this case, sparser decompositions are likely to have better performance, i.e., faster computation and better generalization. Boosting can be interpreted as a greedy forward feature selection procedure to find such sparse solutions.

Although the ideas proposed in this work can be combined with most boosting algorithms, we limit the discussion to adaboost (Freund and Schapire, 1997). This is an algorithm that learns a predictor f by minimizing the risk of (1) when L is the negative exponential of the *margin* $y(x)f(x)$

$$L(y(x), f(x)) = e^{-y(x)f(x)}. \tag{3}$$

This is known as the exponential loss function (Schapire and Singer, 1999).

The boosting algorithms proposed in this paper are inspired by the statistical view of adaboost (Mason et al., 2000; Friedman, 1999). Under this view, each iteration of boosting computes the functional derivatives of the risk along the directions of the weak learners $g_k(x)$, at the current solution $f(x)$. This can be written as

$$\begin{aligned} < \delta\mathcal{R}_E[f], g > &= \frac{d}{d\epsilon}\mathcal{R}_E[f+\epsilon g]\Big|_{\epsilon=0} \\ &= \frac{1}{|S_t|}\sum_i \left[\frac{d}{d\epsilon}e^{-y_i(f(x_i)+\epsilon g(x_i))}\right]_{\epsilon=0} \\ &= -\frac{1}{|S_t|}\sum_i y_i w_i g(x_i), \end{aligned} \tag{4}$$

where $y_i = y(x_i)$ and

$$w_i = w(x_i) = e^{-y_i f(x_i)}, \tag{5}$$

is the weight of example x_i. The latter measures how well x_i is classified by the current predictor $f(x)$. The predictor is then updated by selecting the direction (weak learner) of steepest descent

$$\begin{aligned} g^*(x) &= \arg\max_{g\in\mathbf{G}} < -\delta\mathcal{R}_E[f], g > \\ &= \arg\max_{g\in\mathbf{G}} \frac{1}{|S_t|}\sum_i y_i w_i g(x_i), \end{aligned} \tag{6}$$

and computing the optimal step size along this direction

$$\alpha^* = \arg\min_{\alpha\in\mathbb{R}} \mathcal{R}_E[f+\alpha g^*]. \tag{7}$$

While the optimal step size has a closed form for adaboost (Freund and Schapire, 1997), it can also be found by a line search. The predictor is finally updated according to

$$f(x) = f(x) + \alpha^* g^*(x), \tag{8}$$

and the procedure iterated, as summarized in Algorithm 1.

Algorithm 1 adaboost

```
**Input:** Training set $S_t = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, where $y_i \in \{1, -1\}$ is the class label of example $\mathbf{x}_i$, and number of iterations $N$.
**Initialization:** Set $f(x) = 0$.
**for** $t = 1$ to $N$ **do**
    Compute $< -\delta R_E[f], g >$ for all weak learners using (4).
    Select the best weak learner $g^*(x)$ using (6).
    Find the optimal step size $\alpha^*$ along $g^*(x)$ using (7).
    Update $f(x) = f(x) + \alpha^* g^*(x)$.
**end for**
**Output:** decision rule: $sign[f(x)]$
```

3.2 Cascade Boosting

In this work, we consider the question of whether boosting can be extended to learn a detector cascade. We start by introducing some notation. As shown in Figure 1-a), a classifier cascade is a binary classifier $H(x) \in \mathbf{H}$ implemented as a sequence of classifiers

$$h_i(x) = sgn[f_i(x)] \quad i = 1, \dots, m, \tag{9}$$

where the predictors $f_i(x)$ can be any real functions, e.g., linear combinations of weak learners. The cascade implements the mapping $H : \mathcal{X} \to \{-1, 1\}$ where

$$H(x) = \mathcal{H}^m[h_1, \dots, h_m](x) \quad = \quad \begin{cases} -1 & \text{if } \exists\, k : h_k(x) < 0 \\ +1 & \text{otherwise,} \end{cases} \tag{10}$$

and $\mathcal{H}^m[h_1, \dots, h_m]$ is a *classifier cascading (CC) operator,* i.e., a functional mapping $\mathcal{H}^m : \mathbf{H}^m \to \mathbf{H}$ of the stage classifiers $h_1, \dots, h_m$ into the cascaded classifier H.[1]

Similarly, it is possible to define a cascade predictor $F(x)$ for $H(x)$, i.e., a mapping $F : \mathcal{X} \to \mathbb{R}$ such that

$$H(x) = sign[F(x)], \tag{11}$$

where

$$F(x) = \mathcal{F}^m[f_1, \dots, f_m](x), \tag{12}$$

and $\mathcal{F}^m : \mathbf{F}^m \to \mathbf{F}$ is a *predictor cascading (PC) operator*, i.e., a functional mapping of the stage predictors $f_1, \dots, f_m$ into the cascade predictor F. We will study the structure of this operator in Section 4. For now, we consider the problem of learning a cascade, given that the operator $\mathcal{F}^m$ is known.

To generalize adaboost to this problem it suffices to use the predictor $F(x)$ in the exponential loss of (3) and solve the optimization problem

$$\begin{cases} \min_{m, f_1, \dots f_m} & \mathcal{R}_E[F] = \frac{1}{|S_t|} \sum_i e^{-y_i F(x_i)} \\ \\ s.t: & F(x) = \mathcal{F}^m[f_1, \dots, f_m](x) \\ & \forall i \quad f_i(x) \in span(\mathbf{G}) \end{cases} \tag{13}$$

[1] The notation $\mathcal{H}^m[h_1, \dots, h_m](x)$ should be read as: the value at x of the image of $(h_1, \dots, h_m)$ under operator $\mathcal{H}^m$.

by gradient descent in $span(\mathbf{G})$. The main difference with respect to adaboost is that, since any of the cascade stages can be updated, multiple gradient steps are possible per iteration. The directional gradient for updating the predictor of the k^{th} stage is

$$\begin{aligned}
< \delta\mathcal{R}_E[F], g >_k &= \frac{d}{d\epsilon}\mathcal{R}_E[\mathcal{F}^m[f_1, \dots f_k + \epsilon g, \dots f_m]]\Big|_{\epsilon=0} \\
&= \frac{1}{|S_t|}\sum_i \left[\frac{d}{d\epsilon}e^{-y_i\mathcal{F}^m[f_1,\dots f_k+\epsilon g,\dots f_m](x_i)}\right]_{\epsilon=0} \\
&= \frac{1}{|S_t|}\sum_i \left\{(-y_i)e^{-y_i\mathcal{F}^m[f_1,\dots f_m](x_i)}\left[\frac{d}{d\epsilon}\mathcal{F}^m[f_1, \dots f_k + \epsilon g, \dots f_m]\right]_{\epsilon=0}(x_i)\right\} \\
&= -\frac{1}{|S_t|}\sum_i y_i w(x_i) b_k(x_i) g(x_i), \qquad (14)
\end{aligned}$$

with

$$\begin{aligned}
w(x_i) &= e^{-y_i\mathcal{F}^m[f_1,\dots f_m](x_i)} = e^{-y_iF(x_i)} \qquad (15) \\
b_k(x_i) &= \frac{d}{d\epsilon}\mathcal{F}^m[f_1, \dots f_k + \epsilon g, \dots f_m]\Big|_{\epsilon=0}(x_i). \qquad (16)
\end{aligned}$$

The optimal descent direction for the k^{th} stage is then

$$\begin{aligned}
g_k^* &= \arg\max_{g\in\mathbf{G}} < -\delta\mathcal{R}_E[F], g >_k \\
&= \arg\max_{g\in\mathbf{G}} \frac{1}{|S_t|}\sum_i y_i w(x_i) b_k(x_i) g(x_i), \qquad (17)
\end{aligned}$$

the optimal step size along this direction is

$$\alpha_k^* = \arg\min_{\alpha\in\mathbb{R}} \mathcal{R}_E[\mathcal{F}^m[f_1, .., f_k + \alpha g_k^*, ..f_m]], \qquad (18)$$

and the optimal stage update is

$$f_k(x) = f_k(x) + \alpha^* g^*(x). \qquad (19)$$

The steps of (15), (17), and (19) constitute a functional gradient descent algorithm for learning a detector cascade, which generalizes adaboost. In particular, the weight of (15) generalizes that of (5), reweighing examples by how well the current cascade classifies them. The weak learner selection rule of (17) differs from that of (6) only in that this weight is multiplied by coefficient $b_k(x_i)$. Finally, (19) is an additive update, similar to that of (8). If the structure of the optimal cascade were known, namely how many stages it contains, these steps could be used to generalize Algorithm 1. It would suffice to, at each iteration t, select the stage k such that g_k^* achieves the smallest risk in (18) and update the predictor of that stage. This only has a fundamental difference with respect to adaboost: the introduction of the coefficients $b_k(x_i)$ in the weak learner selection. We will see that the procedure above can also be extended into an algorithm that learns the cascade configuration. Since these extensions depend on the PC operator $\mathcal{F}$ of (12), we start by studying its structure.

4. The Structure of Cascade Predictors

In this section, we derive a general form for $\mathcal{F}^m$. We show that any cascade is compatible with an infinite set of predictors and that these can be computed recursively. This turns out to be important for the efficient implementation of the learning algorithm of the previous section. We next consider a class of PC operators synthesized by recursive application of a two-stage PC operator, denoted the *generator* of the cascade. Two generators are then proposed, from which we derive two new cascade predictor families that generalize the two most common cascade structures in the literature.

4.1 Cascade Predictors

From (10), a classifier cascade implements the logical-AND of the outputs of its stage classifiers, i.e., $\mathcal{H}^m$ is the pointwise logical-AND of $h_1, \ldots, h_m$,

$$\mathcal{H}^m[h_1, \ldots, h_m](x) = h_1(x) \wedge \ldots \wedge h_m(x), \tag{20}$$

where $\wedge$ is the logical-AND operation. Since, from (10)-(12),

$$\mathcal{H}^m[h_1, \ldots, h_m](x) = sgn[\mathcal{F}^m[f_1, \ldots, f_m](x)], \tag{21}$$

it follows from (9) that

$$sign[\mathcal{F}^m[f_1, \ldots, f_m](x)] = sgn[f_1(x)] \wedge \ldots \wedge sgn[f_m(x)]. \tag{22}$$

This holds if and only if

$$\begin{cases} \mathcal{F}^m[f_1, \ldots, f_m](x) < 0 & \text{if } \exists\, k : f_k(x) < 0 \\ \mathcal{F}^m[f_1, \ldots, f_m](x) > 0 & \text{otherwise.} \end{cases} \tag{23}$$

Since (22) holds for any operator with this property, any such $\mathcal{F}^m$ is denoted a pointwise *soft-AND* of its arguments. In summary, while a cascade implements the logical-AND of its stage decisions, the cascade predictor implements a soft-AND of the corresponding stage predictions. Note that there is an infinite number of soft-AND operators which will implement the same logical-AND operator, once thresholded according to (21). This makes the set of cascade predictors much richer than that of cascades.

4.2 Recursive Implementation

For any m, it follows from (20) and the associative property of the logical-AND that

$$\mathcal{H}^m[h_1, \ldots, h_m] \quad = \quad \begin{cases} \mathcal{H}^2[h_1, h_2], & m = 2 \\ \mathcal{H}^2\left[h_1, \mathcal{H}^{m-1}[h_2, \ldots, h_m]\right] & m > 2. \end{cases} \tag{24}$$

A similar decomposition holds for the soft-AND operator of (23), since

$$sgn\left[\mathcal{F}^m[f_1, \ldots, f_m](x)\right] \quad = \quad \begin{cases} sgn\left[\mathcal{F}^2[f_1, f_2](x)\right], & m = 2 \\ sgn\left[\mathcal{F}^2\left[f_1, \mathcal{F}^{m-1}[f_2, \ldots, f_m]\right](x)\right] & m > 2. \end{cases} \tag{25}$$

The main difference between the two recursions is that, while there is only one logical-AND $\mathcal{H}^2[f_1, f_2]$, an infinite set of soft-AND operators $\mathcal{F}^2[f_1, f_2]$ can be used in (25). In fact, it is possible to use a different operator $\mathcal{F}^2$ at each level of the recursion, i.e., replace $\mathcal{F}^2$ by $\mathcal{F}^2_m$, to synthesize all possible sequences of soft-AND operators $\{\mathcal{F}^i\}_{i=2}^m$ for which the left-hand side of (25) is the same. For simplicity, we only consider soft-AND operators of the form of (25) in this work.

The recursions above make it possible to derive a recursive decomposition of both the cascade and the sign of its predictor. In particular, defining

$$H_k(x) \quad = \quad \mathcal{H}^{m-k+1}[h_k, \dots, h_m](x),$$

(24) leads to the *cascade recursion*

$$H_k(x) \quad = \quad \begin{cases} h_m(x), & k = m \\ \mathcal{H}^2\left[h_k, H_{k+1}\right](x), & 1 \leq k < m, \end{cases}$$

with $H_1(x) = H(x)$. Similarly, for any sequence of soft-AND operators $\{\mathcal{F}^i\}_{i=2}^m$ compatible with (25), defining

$$F_k(x) \quad = \quad \mathcal{F}^{m-k+1}[f_k, \dots, f_m](x),$$

leads to the *predictor recursion*

$$sgn[F_k(x)] \quad = \quad \begin{cases} sgn[f_m(x)], & k = m \\ sgn\left[\mathcal{F}^2\left[f_k, F_{k+1}\right](x)\right], & 1 \leq k < m, \end{cases} \tag{26}$$

with $sgn[F_1(x)] = sgn[F(x)]$. Simplifying (26), in the remainder of this work we consider predictors of the form

$$F_k(x) \quad = \quad \begin{cases} f_m(x), & k = m \\ \mathcal{F}^2\left[f_k, F_{k+1}\right](x), & 1 \leq k < m. \end{cases} \tag{27}$$

Since the core of this recursion is the two-stage predictor

$$\mathcal{G}[f_1, f_2] = \mathcal{F}^2[f_1, f_2], \tag{28}$$

this is denoted the *generator* of the cascade. We will show that the two most popular cascade architectures can be derived from two such generators. For each, we will then derive the cascade predictors $F_k(x)$, the cascade boosting weights $w(x_i)$ of (15), and the coefficients $b_k(x_i)$ of (16). We start by defining some notation to be used in these derivations.

4.3 Some Definitions

Some of the computations of the following sections involve derivatives of Heaviside step functions $u(.)$, which are not differentiable. As is common in the neural network literature, this problem is addressed with the sigmoidal approximation

$$u(x) \approx \sigma(x) = \frac{1}{2}(\tanh(\mu x) + 1). \tag{29}$$

The parameter μ controls the sharpness of the sigmoid. This approximation is well known to have the symmetry $\sigma(-x) = 1 - \sigma(x)$ and derivative $\sigma'(x) = 2\mu\sigma(x)\sigma(-x)$. We also introduce the sequence of *cascaded Heaviside functions*

$$\gamma_k(x) \quad = \quad \begin{cases} 1, & k = 1 \\ \prod_{j<k} u[f_j(x)], & k > 1, \end{cases} \tag{30}$$

and *cascaded rectification functions*

$$\xi_k(x) \quad = \quad \begin{cases} 1, & k = 1 \\ \prod_{j<k} f_j(x) u[f_j(x)], & k > 1, \end{cases} \tag{31}$$

where $u(.)$ is the Heaviside step. The former generalize the Heaviside step, in the sense that $\gamma_k(x) = 1$ if $f_j(x) > 0$ for all $j < k$ and $\gamma_k(x) = 0$ otherwise. The latter generalize the half-wave rectifier, in the sense that $\gamma_k(x) = \prod_{j<k} f_j(x)$ if $f_j(x) > 0$ for all $j < k$ and $\gamma_k(x) = 0$ otherwise.

4.4 Last Stage Cascades

The first family of cascade predictors that we consider is derived from the generator

$$\begin{aligned} \mathcal{G}_1[f_1, f_2](x) &= f_1(x)u[-f_1(x)] + u[f_1(x)]f_2(x) \\ &= \begin{cases} f_1(x) & \text{if} \;\; f_1(x) < 0 \\ f_2(x) & \text{if} \;\; f_1(x) \geq 0, \end{cases} \end{aligned} \tag{32}$$

Using (27), the associated predictor recursion is

$$F_k(x) \quad = \quad \begin{cases} f_m(x), & k = m \\ f_k(x)u[-f_k(x)] + u[f_k(x)]F_{k+1}(x), & 1 \leq k < m. \end{cases} \tag{33}$$

The k^{th} stage of the associated cascade passes example x to stage $k+1$ if $f_k(x) \geq 0$. Otherwise, the example is rejected with prediction $f_k(x)$. Hence,

$$\mathcal{F}^m[f_1, \ldots, f_m](x) = \begin{cases} f_j(x) & \text{if} \;\; f_j(x) < 0 \text{ and} \\ & f_i(x) \geq 0 \;\; i = 1, \ldots, j-1 \\ f_m(x) & \text{if} \;\; f_i(x) \geq 0 \;\; i = 1 \ldots, m-1, \end{cases}$$

i.e., the cascade prediction is that of the last stage visited by the example. For this reason, the cascade is denoted a *last-stage cascade*.

This property makes it trivial to compute the weights $w(x)$ of the cascade boosting algorithm, using (15). It suffices to evaluate

$$w(x_i) = e^{-y_i f_{j^*}(x_i)}, \tag{34}$$

where j^* is the smallest k for which $f_k(x_i)$ is negative and $j^* = m$ if there is no such k. The computation of $b_k(x)$ with (16) requires a differentiable form of $\mathcal{F}^m[f_1, \ldots, f_m]$ with respect to f_k. This can be obtained by recursive application of (33), since

$$\begin{aligned}
&\mathcal{F}^m[f_1,\dots,f_m](x) = F_1(x)\\
&\quad= \quad f_1(x)u[-f_1(x)] + u[f_1(x)]F_2(x)\\
&\quad= \quad f_1(x)u[-f_1(x)] + u[f_1(x)]\left\{f_2(x)u[-f_2(x)] + u[f_2(x)]F_3(x)\right\}\\
&\quad= \quad \left[\sum_{i=1}^{k-1} f_i(x)u[-f_i(x)]\prod_{j<i} u[f_j(x)]\right] + F_k(x)\prod_{j<k} u[f_j(x)]\\
&\quad= \quad \left[\sum_{i=1}^{k-1} f_i(x)u[-f_i(x)]\gamma_i(x)\right] + F_k(x)\gamma_k(x) \qquad k = 1\dots m\\
&\quad= \quad \left[\sum_{i=1}^{k-1} f_i(x)u[-f_i(x)]\gamma_i(x)\right] + \gamma_k(x)\left\{f_k(x)u[-f_k(x)] + u[f_k(x)]F_{k+1}(x)\right\} \quad k < m\\
&\quad= \quad \left[\sum_{i=1}^{k-1} f_i(x)u[-f_i(x)]\gamma_i(x)\right] + \gamma_k(x)\left\{f_k(x) + u[f_k(x)][F_{k+1}(x) - f_k(x)]\right\}\\
&\quad\approx \quad \left[\sum_{i=1}^{k-1} f_i(x)u[-f_i(x)]\gamma_i(x)\right] + \gamma_k(x)f_k(x) + \gamma_k(x)\sigma[f_k(x)][F_{k+1}(x) - f_k(x)] \qquad (35)
\end{aligned}$$

where $\gamma_k(x)$ are the cascaded Heaviside functions of (30) and we used the differentiable approximation of (29) in (35). Note that neither the first term on the right-hand side of (35) nor γ_k or F_{k+1} depend on f_k. It follows from (16) that

$$b_k(x) \quad = \quad \begin{cases} \gamma_k(x), & k = m \\ \gamma_k(x)\{1 + 2\mu\sigma[f_k(x)][F_{k+1}(x) - f_k(x)]\}\sigma[-f_k(x)] & 1 \le k < m, \end{cases} \tag{36}$$

where $\sigma(.)$ is defined in (29). Given x, all these quantities can be computed with a sequence of a forward, a backward, and a forward pass through the cascade. The initial forward pass computes $\gamma_k(x)$ for all k according to (30). The backward pass then computes $F_{k+1}(x)$ using (33). The final forward pass computes the weight $w(x)$ and coefficients $b_k(x)$ using (34) and (36). These steps are summarized in Algorithm 2. The procedure resembles the back-propagation algorithm for neural network training (Rumelhart et al., 1968).

4.5 Multiplicative Cascades

The second family of cascade predictors has generator

$$\begin{aligned}
\mathcal{G}_2[f_1, f_2](x) \quad &= \quad f_1(x)u[-f_1(x)] + u[f_1(x)]f_1(x)f_2(x)\\
&= \quad \begin{cases} f_1(x) & \text{if } \ f_1(x) < 0 \\ f_1(x)f_2(x) & \text{if } \ f_1(x) \ge 0. \end{cases}
\end{aligned} \tag{37}$$

Using (27), the associated predictor recursion is

$$F_k(x) \quad = \quad \begin{cases} f_m(x), & k = m \\ f_k(x)u[-f_k(x)] + u[f_k(x)]f_k(x)F_{k+1}(x), & 1 \le k < m \end{cases} \tag{38}$$

Algorithm 2 **Last-stage cascade**

```
Input: Training example $(x, y)$, stage predictors $f_k(x), k = 1, \ldots, m$, sigmoid parameter $\mu$.
Evaluation:
Set $\gamma_1(x) = 1$.
for $k = 2$ to $m$ do
    Set $\gamma_k(x) = \gamma_{k-1}(x)u[f_k(x)]$.
end for
Set $F_m(x) = f_m(x)$.
for $k = m - 1$ to $1$ do
    Set $F_k(x) = f_k(x)u[-f_k(x)] + u[f_k(x)]F_{k+1}(x)$.
end for
Learning:
Set $w(x) = e^{-yf_{j^*}(x)}$ where $j^*$ is the smallest $k$ for which $f_k(x_i) < 0$ and $j^* = m$ if there is no such $k$.
for $k = 1$ to $m - 1$ do
    Set $b_k(x) = \gamma_k(x)\{1 + 2\mu\sigma[f_k(x)][F_{k+1}(x) - f_k(x)]\}\sigma[-f_k(x)]$.
end for
Set $b_k(x) = \gamma_m(x)$.
Output: $w(x), \{F_k(x), b_k(x)\}_{k=1}^m$.
```

and

$$\mathcal{F}^m[f_1, \ldots, f_m](x) = \begin{cases} \prod_{i \le j} f_i(x) & \text{if } \; f_j(x) < 0 \text{ and} \\ & \quad\;\; f_i(x) \ge 0 \quad i = 1..j-1 \\ \prod_{i=1}^{m} f_i(x) & \text{if } \; f_i(x) \ge 0 \quad i = 1..m-1. \end{cases}$$

Hence, the cascade predictor is the product of all stage predictions up-to and including that where the example is rejected. This is denoted a *multiplicative cascade*.

The weights $w(x)$ of the cascade boosting algorithm are

$$w(x_i) = e^{-y_i \prod_{k \le j^*} f_k(x_i)},$$

where j^* is the smallest k for which $f_k(x_i)$ is negative and $j^* = m$ if there is no such k. The computation of $b_k(x)$ with (16) requires a differentiable form of $\mathcal{F}^m[f_1, \ldots, f_m]$ with respect to f_k. This can be obtained by recursive application of (38), since

$$\begin{aligned}
&\mathcal{F}^m[f_1,\ldots,f_m](x) = F_1(x) \\
&\quad= f_1(x)u[-f_1(x)] + u[f_1(x)]f_1(x)F_2(x) \\
&\quad= f_1(x)u[-f_1(x)] + u[f_1(x)]f_1(x)\{f_2(x)u[-f_2(x)] + u[f_2(x)]f_2(x)F_3(x)\} \\
&\quad= \left[\sum_{i=1}^{k-1} f_i(x)u[-f_i(x)]\prod_{j<i} f_j(x)u[f_j(x)]\right] + F_k(x)\prod_{j<k} f_j(x)u[f_j(x)] \\
&\quad= \left[\sum_{i=1}^{k-1} f_i(x)u[-f_i(x)]\xi_i(x)\right] + F_k(x)\xi_k(x) \qquad k = 1\ldots m \\
&\quad= \left[\sum_{i=1}^{k-1} f_i(x)u[-f_i(x)]\xi_i(x)\right] + \xi_k(x)\left\{f_k(x)u[-f_k(x)] + u[f_k(x)]f_k(x)F_{k+1}(x)\right\} \quad k < m \\
&\quad= \left[\sum_{i=1}^{k-1} f_i(x)u[-f_i(x)]\xi_i(x)\right] + \xi_k(x)f_k(x)\left\{1 + u[f_k(x)][F_{k+1}(x) - 1]\right\} \\
&\quad\approx \left[\sum_{i=1}^{k-1} f_i(x)u[-f_i(x)]\xi_i(x)\right] + \xi_k(x)f_k(x)\left\{1 + \sigma[f_k(x)][F_{k+1}(x) - 1]\right\}, \qquad (39)
\end{aligned}$$

where $\xi_i(x)$ are the rectification functions of (31) and we used (38) and the differentiable approximation of (29) in (39). Since neither the first term on the right hand side, ξ_k, or F_{k+1} depend on f_k, it follows from (16) that

$$b_k(x) = \begin{cases} \xi_m(x), & k = m \\ \xi_k(x)\{1 + \sigma[f_k(x)][F_{k+1}(x) - 1]\}\{1 + 2\mu f_k(x)\sigma[-f_k(x)]\} & 1 \le k < m, \end{cases} \qquad (40)$$

where $\sigma(.)$ is defined in (29). Again, these coefficients can be computed with a forward, a backward, and a forward pass through the cascade, which resembles back-propagation, as summarized in Algorithm 3.

5. Learning the Cascade Configuration

Given a cascade configuration, Algorithms 2 or 3, could be combined with the algorithm of Section 3.2 to extend adaboost to the design of last-stage or multiplicative cascades, respectively. However, the cascade configuration is usually not known and must be learned. This consists of determining the number of cascade stages and the number of weak learners per stage.

5.1 Complexity Loss

We start by assuming that the number of cascade stages is known and concentrate on the composition of these stages. So far, we have proposed to simply update, at each boosting iteration, the stage k with the weak learner g_k^* that achieves the smallest risk in (18). While this will produce cascades with good detection accuracy, there is no incentive for the cascade configuration to be efficient, i.e., achieve an optimal trade-off between detection accuracy

Algorithm 3 multiplicative cascade

```
Input: Training example $(x, y)$, stage predictors $f_k(x), k = 1, \ldots, m$, sigmoid parameter $\mu$.
Evaluation:
Set $\xi_1 = 1$.
for $k = 2$ to $m$ do
  Set $\xi_k(x) = \xi_{k-1}(x) f_k(x) u[f_k(x)]$.
end for
Set $F_m(x) = f_m(x)$.
for $k = m - 1$ to 1 do
  Set $F_k(x) = f_k(x) u[-f_k(x)] + u[f_k(x)] f_k(x)\ F_{k+1}(x)$.
end for
Learning:
Set $w(x) = e^{-y \prod_{k \leq j^*} f_k(x)}$ where $j^*$ is the smallest $k$ for which $f_k(x_i) < 0$ and $j^* = m$ if there is no such $k$.
for $k = 1$ to $m - 1$ do
  Set $b_k(x) = \xi_k(x)\{1 + \sigma[f_k(x)][F_{k+1}(x) - 1]\}\{1 + 2\mu f_k(x)\sigma[-f_k(x)]\}$.
end for
Set $b_m(x) = \xi_m(x)$.
Output: $w(x), \{F_k(x), b_k(x)\}_{k=1}^m$.
```

and classification speed. To guarantee such a trade-off it is necessary to search for the most accurate detector under a complexity constraint. This can be done by minimizing the Lagrangian

$$\mathcal{L}[F] \quad = \quad \mathcal{R}_E[F] + \eta \mathcal{R}_C[F], \tag{41}$$

where $F(x)$ and $R_E[F]$ are the cascade predictor and classification risk of (13), respectively,

$$\mathcal{R}_C[F] \quad = \quad E_{X|Y}\{L_C(F, x) | y(x) = -1\} \simeq \frac{1}{|S_t^-|} \sum_{x_i \in S_t^-} L_C(F, x_i),$$

is a complexity risk and η a Lagrange multiplier that determines the trade-off between accuracy and computational complexity. $\mathcal{R}_C[F]$ is the empirical average of a computational loss $L_C(F, x)$, which reflects the number of machine operations required to evaluate $F(x) = \mathcal{F}^m[f_1, \ldots, f_m](x)$, over the set S_t^- of negatives in S_t. The restriction to negative examples is not necessary but common in the classifier cascade literature, where computational complexity is usually defined as the average computation required to reject negative examples. This is mostly because positives are rare and contribute little to the overall computation.

As is the case for the classification risk, where the loss of (3) is an upper bound on the margin and not the margin itself, the computational loss $L_C[F]$ is a surrogate for the computational cost $\mathcal{C}(F, x)$ of evaluating the cascade prediction $F(x)$ for example x. Using the predictor recursions of Section 4.2, this cost can itself be computed recursively. Since, by definition of cascade, example x is either rejected by the predictor f_k of stage k or passed to the remaining stages,

$$\mathcal{C}(F_k, x) \quad = \quad \begin{cases} \Omega(f_k) + u[f_k(x)]\mathcal{C}(F_{k+1}, x), & k < m \\ \Omega(f_m), & k = m, \end{cases} \tag{42}$$

where $F_k(x)$ is as defined in (27) and $\Omega(f_k)$ is the computational cost of evaluating stage k. Defining $\mathcal{C}(F_{m+1}, x) = 0$, it follows that

$$\begin{aligned} \mathcal{C}(F, x) &= \Omega(f_1) + u[f_1(x)]\mathcal{C}(F_2, x) \\ &= \Omega(f_1) + u[f_1(x)][\Omega(f_2) + u[f_2(x)]\mathcal{C}(F_3, x)] \\ &= \left[\sum_{i=1}^{k-1} \Omega(f_i) \prod_{j<i} u[f_j(x)]\right] + \mathcal{C}(F_k, x) \prod_{j<k} u[f_j(x)] \\ &= \left[\sum_{i=1}^{k-1} \Omega(f_i)\gamma_i(x)\right] + \Omega(f_k)\gamma_k(x) + u[f_k(x)]\mathcal{C}(F_{k+1}, x)\gamma_k(x) \\ &= \delta_k(x) + \Omega(f_k)\gamma_k(x) + \theta_k(x)u[f_k(x)], \end{aligned} \tag{43}$$

where $\gamma_i(x)$ are the cascaded Heaviside functions of (30) and

$$\begin{aligned} \delta_k(x) &= \sum_{i=1}^{k-1} \Omega(f_i)\gamma_i(x), \\ \theta_k(x) &= \mathcal{C}(F_{k+1}, x)\gamma_k(x). \end{aligned} \tag{44}$$

This relates the cascade complexity to the complexity of the k^{th} stage, $\Omega(f_k)$. The surrogate computational loss $L_C[F, x]$ is inspired by the surrogate classification loss of adaboost, which upper bounds the zero-one loss $u[-yf(x)]$ by the exponential $e^{-yf(x)}$. Using the bound $u[f(x)] \leq e^{f(x)}$ on (43) leads to

$$L_C[F, x] \quad = \quad \delta_k(x) + \Omega(f_k)\gamma_k(x) + \theta_k(x)e^{f_k(x)},$$

and the computational risk

$$R_C[F] \quad = \quad \frac{1}{|S_t^-|} \sum_{x_i \in S_t^-} \delta_k(x_i) + \Omega(f_k)\gamma_k(x_i) + \theta_k(x_i)e^{f_k(x_i)}. \tag{45}$$

To evaluate this risk, it remains to determine the computational cost $\Omega(f_k)$ of the predictor of the k^{th} cascade stage. Since $f_k(x) = \sum_l \alpha_l g_l(x), g_l \in \mathbf{G}$, is a linear combination of weak learners, we define

$$\Omega(f_k) = \sum_l \Omega(g_l). \tag{46}$$

Let $\mathcal{W}(f_k) \subset \mathbf{G}$ be the set of weak learners, g_l, that appear in (46). In this work, we restrict our attention to the case where all g_l have the same complexity and $\Omega(f_k)$ is proportional to $|\mathcal{W}(f_k)|$. This is the most common scenario in computer vision problems, such as face detection, where all weak learners are thresholded Haar wavelet features (Viola and Jones, 2001) and have similar computational cost. We will, however, account for the fact that there is no cost in the repeated evaluation of a weak learner. For this, $\mathcal{W}(f_k)$ is split into two sets. The first, $\mathcal{O}(f_k)$, contains the weak learners used in some earlier cascade stage $f_j, j \leq k$. Since the outputs of these learners can be kept in memory, they require minimal computation (multiplication by α_l and addition to cumulative sum). The second is the set $\mathcal{N}(f_k)$ of weak learners unused in prior stages. The computational cost of f_k is then

$$\Omega(f_k) = |\mathcal{N}(f_k)| + \lambda|\mathcal{O}(f_k)|, \tag{47}$$

where $\lambda < 1$ is the ratio of computation required to evaluate a used vs. new weak learner. This implies that when updating the k^{th} stage predictor

$$\Omega(f_k + \epsilon g) = \Omega(f_k) + \rho(g, f_k),$$

with

$$\rho(g, f_k) = \begin{cases} \lambda & \text{if } g \in \mathcal{O}(f_k) \\ 1 & \text{if } g \in \mathcal{N}(f_k). \end{cases} \tag{48}$$

5.2 Boosting with Complexity Constraints

Given the computational risk of (45), it is possible to derive a boosting algorithm that accounts for cascade complexity. We start by deriving the steepest descent direction of the Lagrangian of (41), with respect to stage k

$$\begin{aligned} < -\delta\mathcal{L}[F], g >_k &= < -\delta\left(R_E[F] + \eta R_C[F]\right), g >_k \\ &= < -\delta R_E[F], g >_k + \eta < -\delta R_C[F], g >_k . \end{aligned}$$

The first term is given by (14), the second requires the descent direction with respect to the complexity risk $R_C[F]$. Using (45),

$$\begin{aligned} &< \delta R_C[F], g >_k = \frac{d}{d\epsilon} R_C(\mathcal{F}^m[f_1, .., f_k + \epsilon g, ..f_m])\Big|_{\epsilon=0} \\ &= \frac{1}{|S_t^-|}\sum_i y_i^s \frac{d}{d\epsilon} L_C\left[\mathcal{F}^m[f_1, .., f_k + \epsilon g, ..f_m], x_i\right]\Big|_{\epsilon=0} \\ &= \frac{1}{|S_t^-|}\sum_i y_i^s \frac{d}{d\epsilon}\left[\delta_k(x_i) + [\Omega(f_k) + \rho(f_k, g)]\gamma_k(x_i) + \theta_k(x_i) e^{f_k(x_i)+\epsilon g(x_i)}\right]\Big|_{\epsilon=0} \\ &= \frac{1}{|S_t^-|}\sum_i y_i^s \psi_k(x_i)\theta_k(x_i) g(x_i), \end{aligned} \tag{49}$$

where $y_i^s = I(y_i = -1)$, $I(x)$ is the indicator function, $\theta_k(x_i)$ as in (44) and

$$\psi_k(x_i) = e^{f_k(x_i)}. \tag{50}$$

Finally, combining (14) and (49),

$$< -\delta\mathcal{L}[F], g >_k = \sum_i \left(\frac{y_i w(x_i) b_k(x_i)}{|S_t|} - \eta \frac{y_i^s \psi_k(x_i)\theta_k(x_i)}{|S_t^-|}\right) g(x_i), \tag{51}$$

Algorithm 4 BestStageUpdate

```
**Input:** Training set $S_t$, trade-off parameter $\eta$, cascade $[f_1, \ldots, f_m]$, index $k$ of the stage to update, sigmoid parameter $\mu$.
**for** each pair $(x_i, y_i)$ in $S_t$ **do**
    Compute $w(x_i)$, $b_k(x_i)$, $F_k(x_i)$ e.g., using Algorithm 2 for last-stage or Algorithm 3 for multiplicative cascades.
    Compute $\theta_k(x_i)$, $\psi_k(x_i)$ with (44) and (50).
**end for**
Find the best update $(\alpha_k^*, g_k^*(x))$ for the $k^{th}$ stage using (51)-(54).
**Output:** $\alpha_k^*, g_k^*(x)$
```

where $w(x_i) = e^{-y_i F(x_i)}$ and $b_k(x_i)$ is given by (36) for last-stage and by (40) for multiplicative cascades.

It should be noted that, although (51) does not depend on $\rho(f_k, g)$, the complexity of the optimal weak learner g^* affects the computational risk in (45) and thus the magnitude of the steepest descent step. To account for this, we find the best update for f_k in two steps. The first step searches for the best update within $\mathcal{O}(f_k)$ and $\mathcal{N}(f_k)$

$$g_{1,k}^* = \arg \max_{g \in \mathcal{O}(f_k)} < -\delta\mathcal{L}[F], g >_k \quad (52)$$

$$g_{2,k}^* = \arg \max_{g \in \mathcal{N}(f_k)} < -\delta\mathcal{L}[F], g >_k, \quad (53)$$

and computes the corresponding optimal steps sizes

$$\alpha_{j,k}^* = \arg \min_{\alpha \in \mathbb{R}} \mathcal{L}[\mathcal{F}^m[f_1, ..f_k + \alpha g_{j,k}, ..f_m]], \quad (54)$$

for $j = 1, 2$. The second step chooses the update that most reduces $\mathcal{L}[F]$ as the best update for the k^{th} stage. The overall procedure is summarized in Algorithm 4. Using this procedure to cycle through all cascade stage updates within each iteration of the algorithm of Section 3.2 and selecting the one that most reduces $\mathcal{L}[F]$ produces an extension of adaboost for cascade learning that optimizes the trade-off between detection accuracy and complexity.

5.3 Growing a Detector Cascade

So far, we have assumed that the number of cascade stages is known. Since this is usually not the case, there is a need for a procedure that learns this component of the cascade configuration. In this work, we adopt a greedy strategy, where cascade stages are added by the boosting algorithm itself, whenever this leads to a reduction of the risk. It is assumed that a new stage, or predictor g, can only be added at the end of the existing cascade, i.e., transforming a m-stage predictor $\mathcal{F}^m[f_1, \ldots, f_m](x)$ into a $m+1$-stage predictor $\mathcal{F}^{m+1}[f_1, \ldots, f_m, g](x)$. This is consistent with current cascade design practices, where stages are appended to the cascade when certain heuristics are met.

The challenge of a risk-minimizing formulation of this process is to pose the addition of a new stage as a possible gradient step. Recall that, at each iteration of a gradient descent algorithm, the current solution, v^t, is updated by

$$v^{t+1} = v^t + \alpha \overline{v},$$

where $\overline{v}$ is the gradient update and α is step size found by a line search. An immediate consequence is that, if no update is taken in an iteration, i.e., $\alpha = 0$ or $\overline{v} = 0$, the value of the objective function should remain unaltered. For the proposed cascade boosting algorithms this condition is not trivial to guarantee when a new stage is appended to the current cascade. For example, choosing $g(x) = 0$ may change the current solution since, in general,

$$\mathcal{F}^{m+1}[f_1, \dots, f_m, 0](x) \neq \mathcal{F}^m[f_1, \dots, f_m](x).$$

To address this problem, we introduce the concept of *neutral predictors*. A stage predictor $n(x) : \mathcal{X} \to \mathbb{R}$ is neutral for a cascade of predictor $\mathcal{F}^m[f_1, \dots, f_m]$ if and only if

$$\mathcal{F}^{m+1}[f_1, \dots, f_m, n](x) = \mathcal{F}^m[f_1, \dots, f_m](x). \tag{55}$$

If such a neutral predictor exists, then it is possible to grow a cascade by defining the new stage as

$$f_{m+1}(x) = n(x) + g(x),$$

where $g(x)$ is the best update found by gradient descent. In this case, it follows from (55) that a step of $g(x) = 0$ will leave the cascade risk unaltered. Given a cascade generator, a predictor n that satisfies (55) can usually be found with (28), i.e., it suffices that n satisfies

$$f_m(x) = \mathcal{G}[f_m, n](x), \tag{56}$$

where $\mathcal{G}$ is the generator that defines the PC operator $\mathcal{F}^m$. For example, from (32), the neutral predictor of a last-stage cascade must satisfy

$$f_m(x) = f_m(x)u[-f_m(x)] + u[f_m(x)]n(x),$$

a condition met by

$$n(x) = f_m(x). \tag{57}$$

Similarly, from (37), the neutral predictor of a multiplicative cascade must satisfy

$$f_m(x) = f_m(x)u[-f_m(x)] + u[f_m(x)]f_m(x)n(x),$$

which is met by

$$n(x) = 1. \tag{58}$$

These neutral predictors are also computationally efficient. In fact, (57) and (58) add no computation to the evaluation of predictor $f_{m+1}(x)$, i.e., to the computation of $g(x)$ itself. This is obvious for (58) which is a constant, and follows from the fact that $f_m(x)$ has already been computed in stage m for (57). This computation can simply be reused at stage $m + 1$ with no additional cost. Hence, for both models

$$\mathcal{C}(\mathcal{F}^{m+1}[f_1, \dots, f_m, n], x) = \mathcal{C}(\mathcal{F}^m[f_1, \dots, f_m], x),$$

and

$$\mathcal{L}[\mathcal{F}^{m+1}[f_1, \dots, f_m, n]] = \mathcal{L}[\mathcal{F}^m[f_1, \dots, f_m]].$$

In summary, the addition of stages *does not* require special treatment in the proposed cascade learning framework. It suffices to append a neutral predictor to the cascade and find the best update for this new stage. If this reduces the objective function of (41) further than updating other stages, the new stage is *automatically* created and appended to the cascade. In this way, the cascade grows organically, as a side effect of the risk optimization, and there is no need for heuristics.

Algorithm 5 FCBoost

```
Input: Training set $S = \{(\mathbf{x}_1, y_1) \ldots, (\mathbf{x}_n, y_n)\}$, trade-off parameter $\eta$, sigmoid parameter $\mu$, and number of iterations $N$.
Initialization: Set $m = 0$ and $f_1(x) = n(x)$, e.g., using (57) for last-stage and (58) for multiplicative cascade.
for $t = 1$ to $N$ do
    for $k = 1$ to $m$ do
        $(\alpha^*_k, g^*_k) = \text{BestStageUpdate}(S, \eta, [f_1, ...f_m], k, \mu)$.
    end for
    $(\alpha^*_{m+1}, g^*_{m+1}) = \text{BestStageUpdate}(S, \eta, [f_1, ...f_{m+1}], m + 1, \mu)$.
    for $k = 1$ to $m$ do
        Set $\hat{\mathcal{L}}(k) = \mathcal{L}\left[\mathcal{F}^m(f_1, .., f_k + \alpha^*_k g^*_k, .., f_m)\right]$ using (41).
    end for
    Set $\hat{\mathcal{L}}(m + 1) = \mathcal{L}\left[\mathcal{F}^{m+1}(f_1, .., f_m, f_{m+1} + \alpha^*_{m+1} g^*_{m+1}(x))\right]$ using (41).
    Find $k^* = \arg\min_{k \in \{1,...,m+1\}} \hat{\mathcal{L}}(k)$.
    Set $f_{k^*} = f_{k^*} + \alpha^*_{k^*} g^*_{k^*}$.
    if $k^* = m + 1$ then
        Set $m = m + 1$ .
        Set $f_{m+1}(x) = n(x)$.
    end if
end for
Output: decision rule: $sgn[\mathcal{F}^m(f_1, \ldots, f_m)]$.
```

6. The FCBoost Cascade Learning Algorithm

In this section, we combine the contributions from the previous sections into the *Fast Cascade Boosting* (FCBoost) algorithm, discuss its connections with the previous literature and some interesting properties.

6.1 FCBoost

FCBoost is initialized with a neutral predictor. At each iteration, it finds the best update $g^*_k(x)$ for each of the cascade stages and the best stage to add at the end of the cascade. It then selects the stage k^* whose update $g^*_{k^*}(x)$ most reduces the Lagrangian $\mathcal{L}[F]$. If k^* is the newly added stage, a new stage is created and appended to the cascade. The procedure is summarized in Algorithm 5. Note that the only parameters are the multiplier η of (41), which encodes the relative importance of cascade speed vs. accuracy for the cascade designer, and the sigmoid parameter μ that controls the smoothness of the Heaviside approximation. In our implementation we always use $\mu = 5$. Given these parameters, FCBoost will automatically determine both the cascade configuration (number of stages and number of weak learners per stage) and the predictor of each stage, so as to optimize the trade-off between detection accuracy and complexity which is specified through η.

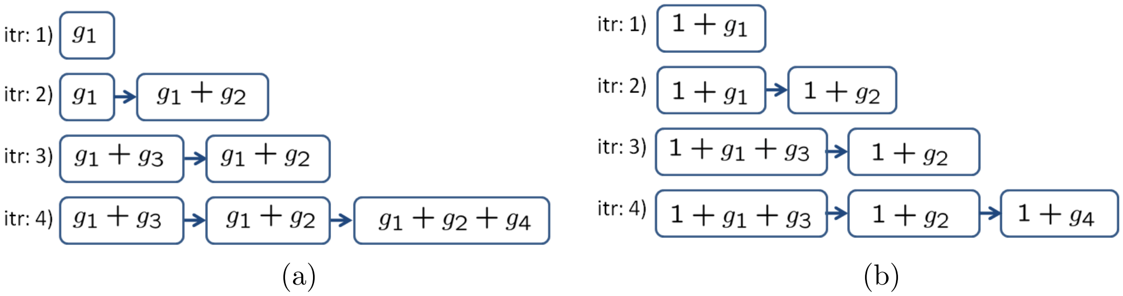


Figure 2: Illustration of the different configurations produced by identical steps of (a) last-stage and (b) multiplicative cascade learning.

6.2 Connections to the Previous Cascade Learning Literature

FCBoost supports a large variety of cascade structures. The cascade structure is defined by the generator $\mathcal{G}$ of (28), since this determines the neutral predictor $n(x)$, according to (56), and consequently how the cascade grows as boosting progresses. The two cascade predictors used in this work, last-stage and multiplicative, cover the two predominant cascade structures in the literature. The first is the *independent stage* (IS) structure, (Viola and Jones, 2001). In this structure stage predictors are designed independently,[2] in the sense that the learning of f_k starts from an empty predictor which is irrespective of the composition of the previous stages, $f_j, j < k$. The second structure is the *embedded stage* (ES) structure where predictors of consecutive stages are related by

$$f_{k+1}(x) = f_k(x) + \mathbf{w}(x),$$

and $\mathbf{w}(x)$ is a single or linear combination of weak learners (Xiao et al., 2003). Under this structure, each stage predictor contains the predictor of the previous stage, which is augmented with some weak learners.

The connection between these structures and the models proposed in this paper can be understood by considering the neutral predictors of the latter. For multiplicative cascades, it follows from (58) that

$$f_{m+1}(x) = 1 + \alpha g(x),$$

and there is no dependence between consecutive stages. Hence, multiplicative cascades have the IS structure. For last stage cascades, it follows from (57) that

$$f_{m+1}(x) = f_m(x) + \alpha g(x).$$

If FCBoost always updates the last two stages, this produces a cascade with the ES structure. Since FCBoost is free to update any stage, it can produce more general cascades, i.e., a superset of the set of cascades with the ES structure.

It is interesting that two predictors with the very similar generators of (32) and (37) produce very different cascade structures. This is illustrated in Figure 2, where we consider the cascades resulting from the following sequence of operations:

[2] Note that the predictors are always *statistically* dependent, since the role of h_{i+1} is to classify examples not rejected by h_i.

- **iteration 1:** start form an empty classifier, create first stage.
- **iteration 2:** add a new stage.
- **iteration 3:** update first stage.
- **iteration 4:** add a new stage.

Note that while the last-stage cascade of a) has substantial weak learner sharing across stages, this is not true for the multiplicative cascade of b), which is similar to the Viola and Jones cascade (Viola and Jones, 2001).

6.3 Properties

Beyond these connections to the literature, FCBoost has various interesting properties as a cascade boosting algorithm. First, its example weighing is very similar to that of adaboost (Freund and Schapire, 1997). A comparison of (5) and (15) shows that FCBoost reweights examples by how well they are classified by the current cascade. As in adaboost, this is measured by the classification margin, but now with respect to the cascade predictor, F, (margin yF) rather than a simple predictor f (margin yf). Second, the weak learner selection rule of FCBoost is very similar to that of adaboost. While in (6) adaboost selects the weak learner g that maximizes

$$\frac{1}{|S_t|}\sum_i y_i w_i g(x_i),$$

in (52)-(53) FCBoost selects the stage k and weak learner g that maximize

$$\sum_i \left(\frac{y_i w(x_i) b_k(x_i)}{|S_t|} - \eta \frac{y_i^s \psi_k(x_i) \theta_k(x_i)}{|S_t^-|} \right) g(x_i). \tag{59}$$

When $\eta = 0$, the only significant difference is the inclusion of $b_k(x_i)$ in (59). To understand the role of this term note that, from (36) and (40), $b_k(x_i) = 0$ whenever $\gamma_k(x_i) = 0$ in (30), and $\xi_k(x_i) = 0$ in (31). This implies that there is at least one stage $j < k$ such that $f_j(x_i) < 0$, i.e., where x_i is rejected. When this holds, $b_k(x_i) = 0$ prevents x_i from influencing the update of $f_k(x)$. This is sensible: since x_i will not reach the k^{th} stage, it should not affect its learning. Hence, the coefficients $b_k(x_i)$ can be seen as *gating coefficients*, which prevent examples rejected by earlier stages from affecting the learning of stage k. If $\eta \neq 0$, a similar role is played by $\theta_k(x_i)$ in the second term of (59) since, from (44), $\theta_k(x_i) = 0$ whenever $\gamma_k(x_i) = 0$. Thus, if x_i is rejected by a stage $j < k$, its processing complexity is not considered for any stage posterior to j. Due to the gating coefficients $b_k(x_i)$ and $\theta_k(x_i)$, FCBoost emulates the bootstrapping procedure commonly used in cascade design. This is a procedure that eliminates the examples rejected by each stage from the training set of subsequent stages. These examples are replaced with new false positives (Viola and Jones, 2001; Sung and Poggio, 1998). While FCBoost emulates “example discarding” with the gating coefficients $b_k(x_i)$ and $\theta_k(x_i)$, it does not seek new false positives. This still requires the “training set augmentation” of bootstrapping.

A third interesting property of FCBoost is the complexity penalty (second term) of (59). From (44) and (50) this is, up to constants,

$$-\,y_i^s \gamma_k(x_i) e^{f_k(x_i)} \mathcal{C}(F_{k+1}, x_i) g(x_i).$$

Given example x_i and cascade stage k, all factors in this product have a meaningful interpretation. First, since $y_i^s \gamma_k(x_i)$ is non-zero only for negative examples which have not been rejected by earlier cascade stages ($j < k$), it acts as a selector of the false-positives that reach stage k. Second, since $f_k(x_i)$ measures how deeply x_i penetrates the positive side of the stage k classification boundary, $e^{f_k(x_i)}$ is large for the false-positives that stage k confidently assigns to the positive class. Third, since $\mathcal{C}(F_{k+1}, x_i)$ is the complexity of processing x_i by the stages beyond k, it measures how deeply x_i penetrates the cascade, if not rejected by stage k. Finally, $g(x_i)$ is the label given to x_i by weak learner $g(x)$. Since only $g(x_i)$ can be negative, the product is maximized when $g(x_i) = -1$, $\gamma_k(x_i) = 1$ and $f_k(x_i)$ and $\mathcal{C}(F_{k+1}, x_i)$ are as large as possible. Hence, the best weak learner is that which, on average, declares as negatives the examples which 1) are false-positives of the earlier stages, 2) are most confidently accepted as false-positives by the current stage, and 3) penetrate the cascade most deeply beyond this stage. This is intuitive, in the sense that it encourages the selection of the weak learner that most contradicts the current cascade on its most costly mistakes.

In summary, FCBoost is a generalization of adaboost with similar example weighting, gating coefficients that guarantee consistency with the cascade structure, and a cost function that accounts for classifier complexity. This encourages the selection of weak learners that correct the false-positives of greatest computational cost. It should be mentioned that while we have used adaboost to derive FCBoost, similar algorithms could be derived from other forms of boosting, e.g., logitboost, gentle boost (Friedman et al., 1998), KLBoost (Liu and Shum, 2003) or float boost (Li and Zhang, 2004). This would amount to replacing the exponential loss, (3), with other loss functions. While the resulting algorithms would be different, the fundamental properties (example reweighing, additive updates, gating coefficients) would not. We next exploit this to develop a cost-sensitive extension of FCBoost.

6.4 Cost-Sensitive FCBoost

While positive examples rejected by a cascade stage cannot be recovered by subsequent stages, the cascade false positive rate can always be reduced through addition of stages. Hence, in cascade learning, maintaining a high detection rate across stages is more critical than maintaining a low false positive rate. This is difficult to guarantee with the risk of (1), which is an upper bound on the error rate, treating misses and false positives equally. Several approaches have been proposed to enforce asymmetry during cascade learning. One possibility is to manipulate the thresholds of the various detector stages to guarantee the desired detection rate (Viola and Jones, 2001; Sochman and Matas, 2005; Luo, 2005). This is usually sub-optimal, since boosting predictors are not well calibrated outside a small neighborhood of the classification boundary (Mease and Wyner, 2008). Threshold tuning merely changes the location of the boundary and can perform poorly (Masnadi-Shirazi and Vasconcelos, 2010). An alternative is to use cost sensitive boosting algorithms (Viola and Jones, 2002;

Masnadi-Shirazi and Vasconcelos, 2007), derived from asymmetric losses that weigh miss-detections more than false-positives, optimizing the cost-sensitive boundary directly. This usually outperforms threshold tuning.

In this work we adopt the cost sensitive risk of

$$\begin{aligned} R_E^c(f) &= \frac{C}{|S_t^+|}\sum_{x_i \in S_t^+} e^{-y_i f(x_i)} + \frac{1-C}{|S_t^-|}\sum_{x_i \in S_t^-} e^{-y_i f(x_i)} \\ &= \sum_{x_i \in S_t} y_i^c e^{-y_i f(x_i)}, \end{aligned} \tag{60}$$

where $C \in [0,1]$ is a cost factor,

$$y_i^c = \frac{C}{|S_t^+|} I(y_i = 1) + \frac{1-C}{|S_t^-|} I(y_i = -1),$$

$I(.)$ the indicator function, and the relative importance of positive vs. negative examples is determined by the ratio $\frac{C}{1-C}$ (Viola and Jones, 2002). This leads to the cost-sensitive Lagrangian

$$\mathcal{L}^c[F] = R_E^c[F] + R_C[F]. \tag{61}$$

A derivation similar to that of (14) can be used to show that

$$< \delta R_E^c[F], g >_k = -\sum_i y_i y_i^c w(x_i) b_k(x_i) g(x_i), \tag{62}$$

where $w(x_i) = e^{-y_i F(x_i)}$ and $b_k(x_i)$ is given by (36) for last-stage and by (40) for multiplicative cascades. Finally, combining (61), (62), and (49),

$$< -\delta \mathcal{L}^c[F], g >_k \;=\; \sum_i \left(y_i y_i^c w(x_i) b_k(x_i) - \eta \frac{y_i^s \psi_k(x_i) \theta_k(x_i)}{|S_t^-|} \right) g(x_i). \tag{63}$$

The cost-sensitive version of FCBoost replaces (51) with (63) in (52)-(53) and $\mathcal{L}$ by $\mathcal{L}^c$ in (54).

6.5 Open Issues

One subtle difference between adaboost and FCBoost, with $\eta = 0$, is the feasible set of the underlying optimization problems. Rewriting the FCBoost problem of (13) as

$$\left\{ \begin{array}{ll} \min_f & R_E[f] \\ s.t: & f \in \Omega_{\mathbf{G}}, \end{array} \right. \tag{64}$$

where

$$\Omega_{\mathbf{G}} = \{ f | \exists f_1, ... f_m \in \mathbf{G} \quad \text{such that} \quad f(x) = \mathcal{F}^m[f_1, \ldots, f_m](x) \quad \forall x \}.$$

and comparing (64) to (2), the two problems differ in their feasible sets, $span(\mathbf{G})$ for adaboost vs. $\Omega_{\mathbf{G}}$ for FCBoost. Since any $\hat{f} \in span(\mathbf{G})$ is equivalent to a one-stage cascaded predictor, it follows that $\hat{f} \in \Omega_{\mathbf{G}}$ and

$$span(\mathbf{G}) \subset \Omega_{\mathbf{G}}.$$

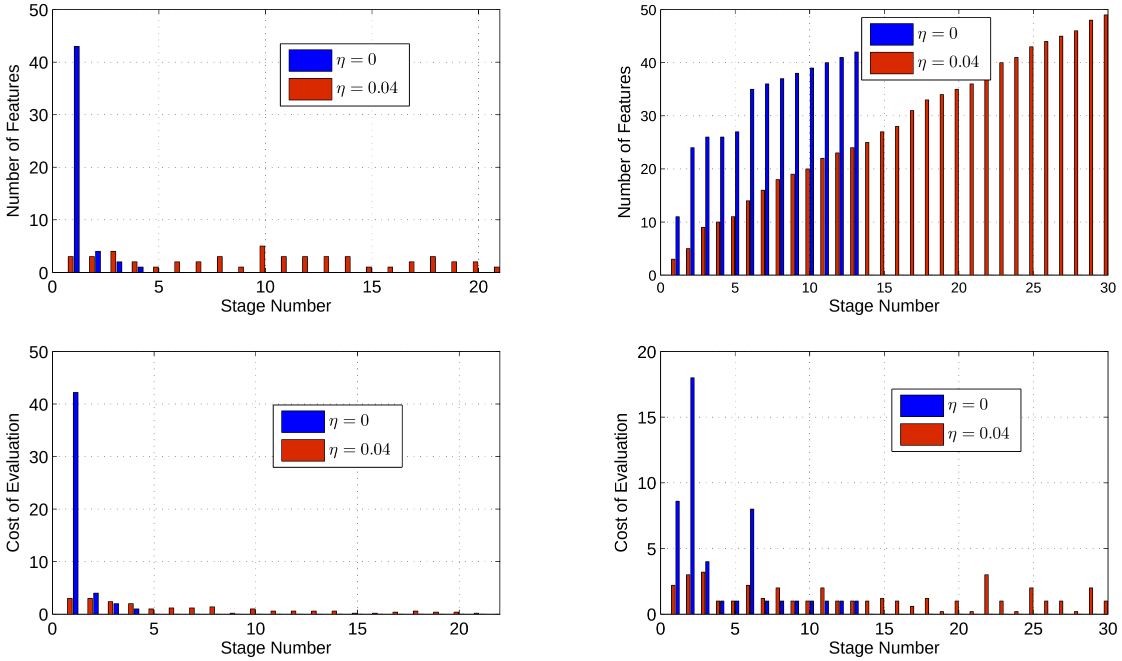
Hence, the feasible set of FCBoost is larger than that of adaboost, and FCBoost can, in principle, find detectors of lower risk. Hence, all generalization guarantees of adaboost hold, in principle, for cascades learned with FCBoost. There is, however, one significant difference. Since $span(\mathbf{G})$ is a convex set, the optimization problem of (2) is convex whenever $R_E(f)$ is a convex function of f. This is the case for the adaboost risk, and adaboost is thus guaranteed to converge to a global minimum. However, since $\Omega_{\mathbf{G}}$ can be a non-convex set, no such guarantees exist for FCBoost. Hence, FCBoost can converge to a local minimum. We illustrate this with an example in Section 7.1. In general, the convexity of $\Omega_{\mathbf{G}}$ depends on the PC operator $\mathcal{F}^m$ and the set of weak learners $\mathbf{G}$. There is currently little understanding on what conditions are necessary to guarantee convexity.

7. Evaluation

In this section, we report on several experiments conducted to evaluate FCBoost. We start with a set of experiments designed to illustrate the properties of the algorithm. We then report results on its use to build face and pedestrian detectors with state-of-the-art performance in terms of detection accuracy and complexity. In all cases, the training set for face detection contained $4,500$ faces (along with their flipped replicas) and $9,000$ negative examples, of size 24×24 pixels, while pedestrian detection relied on a training set of $2,347$ positive and $2,000$ negatives examples, of size 72×30, from the Caltech Pedestrian data set (Dollár et al., 2012). All weak learners were decision-stumps on Haar wavelets (Viola and Jones, 2001).

7.1 Effect of η

We started by studying the impact of the Lagrange multiplier η, of (41), on the accuracy vs. complexity performance of FCBoost cascades. The test set consisted of 832 faces (along with their flipped replicas) and $1,664$ negatives. All detectors were trained for 50 iterations. The unit computational cost was set to the cost of evaluating a new Haar feature. This resulted in a cost of $\frac{1}{5}$ units for feature recycling, i.e., $\lambda = \frac{1}{5}$ in (47). Figure 3 quantifies the structure of the cascades learned by FCBoost with $\eta = 0$ and $\eta = 0.04$: multiplicative in a) and last-stage in b). The top plots summarize the number of features assigned to each cascade stage, and those at the bottom the computational cost per stage. Note that since, from (57), the neutral predictor of the last-stage cascade is its last stage, each of the last-stage cascade stages benefits from the features evaluated in the previous stages. Hence, as shown in the top plot of Figure 3-b, the number of weak learners per stage is monotonically increasing. However, because most features are recycled, the cost is still dominated by the early stages, when $\eta = 0$. With respect to the impact of η, its is clear that, for both structures, a small η produces short cascades whose early stages contain many weak learners. On the other hand, a large η leads to much deeper cascades, and a more uniform distribution of weak learners and computation. This is sensible, since larger η place more emphasis on computational efficiency and this requires that the early stages, which tend to be evaluated for most examples, be very efficient. Hence, long cascades with a few weak learners per stage tend to be computationally more efficient than short cascades with many learners per stage.



(a)

(b)

Figure 3: Number of features (top) and computational cost (bottom) per stage of an FCBoost cascade: (a) multiplicative, (b) last-stage.

The accuracy vs. complexity trade-off of these cascades was compared to those of a non-cascaded adaboost detector and a cascade of embedded stages derived from this detector (Masnadi-Shirazi and Vasconcelos, 2007). This converts the detector into a cascade by inserting a rejection point per weak learner. The resulting cascade has embedded stages which add a single weak learner to their predecessors and is equivalent to the chain boost cascade (Xiao et al., 2003). Figure 4 depicts the trade-off between computation and accuracy of adaboost, chain boost, and FCBoost cascades with $\eta \in [0, 0.04]$. The left-most (right-most) point on the FCBoost curves corresponds to $\eta = 0$ ($\eta = 0.04$). adaboost and ChainBoost points were obtained by limiting the number of weak learners, with a single weak learner (full detector) for the right-most (left-most) point. Several observations can be made from the figure. First, as expected, increasing the trade-off parameter η produces FCBoost cascades with less computation and higher error. Second, FCBoost has a better trade-off between complexity and accuracy (curves closer to the origin). Third, among FCBoost models, last-stage cascades have uniformly better trade-off than their multiplicative counterparts. Since last-stage are generalized embedded cascades, this confirms previous reports on the advantages of embedded over independent stages (Pham et al., 2008; Xiao et al., 2003). Finally, it is interesting to note that, when $\eta = 0$, the Lagrangian of (41) is equivalent to the adaboost risk, i.e., FCBoost and adaboost minimize the same objective. However, due to their different feasible sets, they can learn very different detectors (see Section 6.5). While

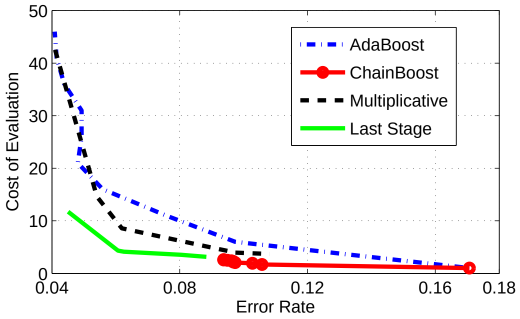


Figure 4: Computational cost vs. error rate of the detectors learned with adaboost, chain boost, and FCBoost with the last-stage and multiplicative structures.

	adaboost	FCBoost+last-stage	FCBoost+multiplicative
Err. rate	4.03%	4.51%	4.15%
Eval. cost	50	11.74	42.54

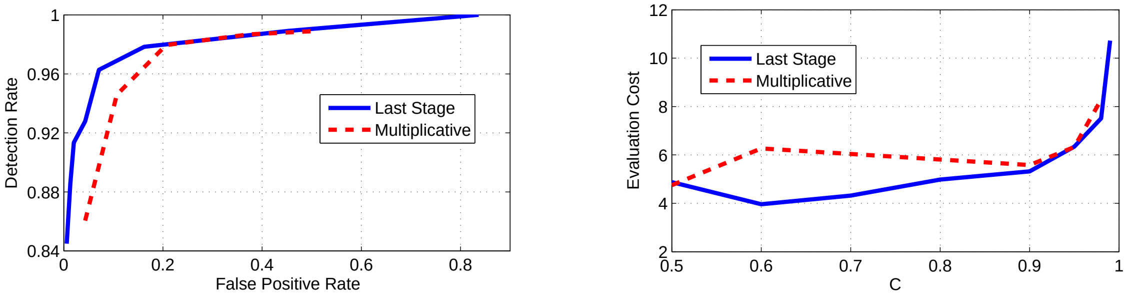
Table 1: Performance comparison between adaboost and FCBoost, for $\eta = 0$.

the larger feasible set of FCBoost suggests that it should produce detectors of smaller risk than adaboost, this did not happen in our experiments.

Table 1 summarizes the error and cost of adaboost and the two FCBoost methods for $\eta = 0$. Note that the adaboost detector has a slightly lower error. The weaker accuracy of the FCBoost detectors suggests that the latter does get trapped in local minima. This is, in fact, intuitive as the decision to add a cascade stage makes it impossible for the gradient descent procedure to revert back to a non-cascaded detector. By making such a decision, FCBoost can compromise the global optimality of its solution, if the global optimum is a non-cascaded detector. Interestingly, FCBoost sometimes decides to add stages even when $\eta = 0$ (see Figure 3). As shown in Table 1, this leads to a slightly more error-prone but much more efficient detector than adaboost. In summary, even without pressure to minimize complexity ($\eta = 0$), FCBoost may trade-off error for complexity. This may be desirable or not, depending on the application. In the experiment of Table 1, FCBoost seems to make sensible choices. For the last-stage structure, it trades a small increase in error (0.48%) for a large decrease in computation (76.5%). For the multiplicative structure, it trades-off a very small increase in error (0.12%) for a moderate (16%) decrease in computation.

7.2 Cost-Sensitive FCBoost

We next consider the combination of FCBoost and the cost sensitive risk of (60). Since the advantages of cost-sensitive boosting over threshold tuning are now well established (Viola and Jones, 2002; Masnadi-Shirazi and Vasconcelos, 2010), we limit the discussion to the effect of the cost factor C on the behavior of FCBoost cascades. Cascaded face detectors were learned for cost factors $C \in \{0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99\}$. Figure 5 a) presents the trade-off between detection and false positive rate for last-stage and multiplicative



(a) (b)

Figure 5: Performance of cascades learned with cost-sensitive FCBoost, using different cost factors C. (a) ROC curves, (b) computational complexity.

cascades. In both cases, the leftmost (rightmost) point corresponds to $C = 0.5$ ($C = 0.99$). Figure 5 b) presents the equivalent plot for computational cost. Several observations can be made. First, as expected, larger cost factors C produce detectors of higher detection and higher false-positive rate. Second, they lead to cascades of higher complexity. This is intuitive since, for large cost factors, FCBoost aims for a high detection rate and is very conservative about rejecting examples. Hence, many negatives penetrate deep into the cascade, and computation increases. Third, comparing the curves of the last-stage and multiplicative cascades, the former again has better performance. In particular, last-stage cascades combine higher ROC curves in Figure 5 a) with lower computational cost in Figure 5 b).

7.3 Face and Pedestrian Detection

Over the last decade, there has been significant interest in the problem of real-time object detection from video streams. In particular, the sub-problems of face and pedestrian detection have been the focus of extensive research, due to the demand for face detection in low-power consumer electronics (e.g., cameras or smart-phones) and pedestrian detection in intelligent vehicles. In this section, we compare the performance of FCBoost cascades with those learned by several state of the art methods in the face and pedestrian detection literatures.

We start with face detection, where cascaded detectors have become predominant, comparing FCBoost to the method of Viola and Jones (VJ) (Viola and Jones, 2001), Wald boost (Sochman and Matas, 2005) and multi-exit (Pham et al., 2008). Since extensive results on these and other methods are available on the MIT-CMU test set, all detectors were evaluated on this data set. The methods above have been shown to outperform a number of other cascade learning algorithms (Pham et al., 2008) and, to the best of our knowledge, hold the best results in this data set. In all cases, the target detection rate was set to $D_T = 95\%$. For Wald boost, multi-exit, and VJ, the training set was bootstrapped when a new stage was added to the cascade, for FCBoost when the false positive rate dropped below 95%. For VJ and multi-exit cascades, which require the specification of the number of cascade stages and a target false-positive and detection rate per stage, we used 20 stages,

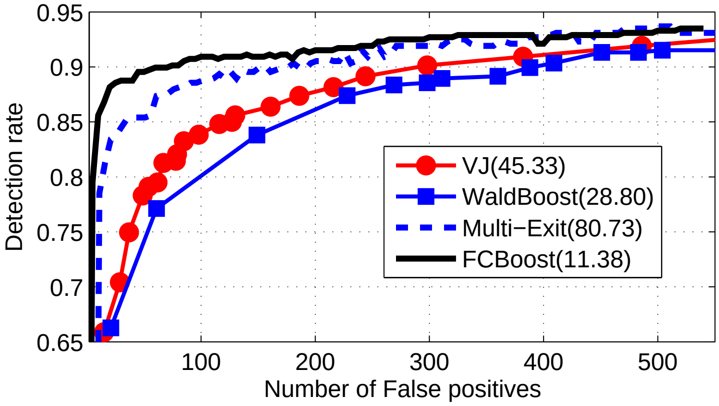


Figure 6: ROCs of various face detectors on MIT-CMU. The number in the legend is the average evaluation cost, i.e., average number of features evaluated per sub-window.

and the popular strategy of setting the false positive rate to 50% and the detection rate to $D_T^{\frac{1}{20}}$. For FCBoost we used a last-stage cascade, since this structure achieved the best balance between accuracy and speed in the previous experiment. We did not attempt to optimize η, simply using $\eta = 0.02$. The cost factor C was initialized with $C = 0.99$. If after a boosting update the cascade did not meet the detection rate, C was increased to

$$C_{new} = \frac{C_{old} + 1}{2}. \tag{65}$$

This placed more emphasis on avoiding misses than false positives, and was repeated until the updated cascade satisfied the rate constraint. The final value of C was used as the initial value for the next boosting update.

Figure 6 show the ROCs of all detectors. The average evaluation cost, i.e., average number of features evaluated per sub-window, is shown in the legend for each method. Note that the FCBoost cascade is simultaneously more accurate and faster than those of all other methods. For example, at 100 false positives, FCBoost has a detection rate of 91% as opposed to 88% for multi-exit, 83% for VJ, and 80% for Wald boost. With regards to computation, FCBoost is 7.1, 4, and 2.5 times faster than multi-exit, VJ, and Wald boost, respectively. Overall, when compared to the FCBoost cascade, the closest cascade in terms of detection rate (multi-exit, 3% drop) is significantly slower (7 times) and the closest cascade in terms of detection speed (Wald boost, 2.5 times slower) has a very poor detection rate (11% smaller).

We next considered the problem of pedestrian detection, comparing results to a large set of state-of-the-art pedestrian detectors on the Caltech Pedestrian data set (Dollár et al., 2012). In this literature, it is well known that a good representation for pedestrians must account for both edge orientation and color (Dalal and Triggs, 2005; Dollár et al., 2009). Similarly to Dollar *et al.* (Dollár et al., 2009), we adopted an image representation based on a 10 channel decomposition. This included 3 color channels (YUV color space), 6 gradient orientation channels, and a gradient magnitude channel. In all other aspects, the cascade

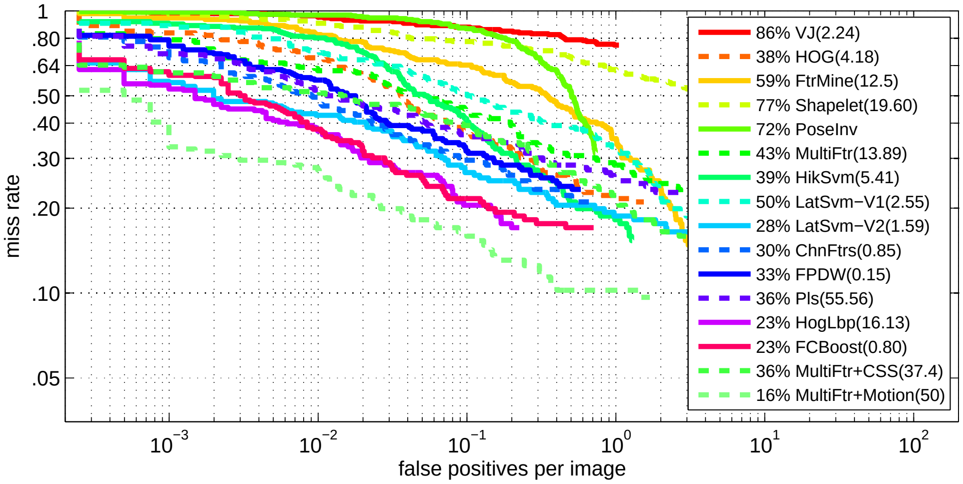


Figure 7: Accuracy curves and complexity of various pedestrian detectors on the Caltech data set. Legend: (left) miss rates at 0.1 FPPI, (right) average time, in seconds, required to process 480 × 640 frame.

architecture was as before, e.g., using Haar wavelet features and decision stumps as weak learners, the previously used values for parameters D_T, and η, etc. When compared to the face detection experiments, the only difference is that the set of weak learners was replicated for each channel. At each iteration, FCBoost chose the best weak learner and the best channel to add to the cascade predictor. The performance of the FCBoost cascade was evaluated with the toolbox of (Dollár et al., 2012). Figure 7 compares its complexity and curve of miss-detection rate vs number of false positives per image (FPPI) to those of a number of recent pedestrian detectors. The comparison was restricted to the popular near scale-large setting, which evaluates the detection of pedestrians with more than 100 pixels in height. The numbers shown in the left of the legend summarize the detection performance by the miss rate at 0.1 FPPI. The numbers shown in the right indicated the average time, in seconds, required for processing a 480 × 640 video frame. Note that the evaluation is *not* restricted to fast detectors, including the most popular architectures for object detection in computer vision, such as the HOG detector (Dalal and Triggs, 2005) or the latent SVM (Felzenszwalb et al., 2010). For more information on the curves and other methods the reader is referred to Dollár et al. (2012).

Two sets of conclusions can be derived from these results. First, they confirm the observation that the FCBoost cascade significantly outperforms previous cascaded detectors. A direct comparison is in fact possible against the ChnFtrs method (Dollár et al., 2009). This work introduced the multi channel features that we adopt but uses the SoftCascade algorithm (Bourdev and Brandt, 2005) for cascade learning. The resulting detector is among the top methods on this data set, missing 30% of the pedestrians at 0.1 FPPI and using 0.85 seconds to process a frame. Nevertheless, the FCBoost cascade has substantially better accuracy, missing only 23% of the pedestrians at 0.1 FPPI, and requires less time (a 6% speed up). Second, the results of Figure 7 show that the FCBoost cascade is one of the most accurate pedestrian detectors in the literature, and significantly faster than the detectors of comparable accuracy. In fact, only two detectors have been reported to achieve equivalent or lower miss rates. The Hog-Lbp detector (Wang et al., 2009) has the same miss rate (23% at 0.1 FPPI) but is 20 times slower. The MultiFtr+Motion (Walk et al., 2010) detector has a smaller miss rate of 16% (at 0.1 FPPI) but is 62 times slower (almost 1 minute per frame). The inclusion of this method in Figure 7 is somewhat unfair, since it is the only approach that exploits motion features. All other detectors, including the FCBoost cascade, operate on single-frames. We did not investigate the impact of adding motion features to FCBoost. Finally, it should be noted that the FCBoost cascade could be enhanced with various computational speed ups proposed in the design of the FPDW detector (Dollár et al., 2010). This is basically a fast version of the ChnFtrs detector, using several image processing speed-ups to reduce the time necessary to produce the image channels on which the classifier operates. These speed-ups lead to a significant increase in speed (0.15 vs 0.85 seconds) at a marginal cost in terms of detection accuracy (33% vs. 30% miss rate at 0.1 FPPI). Since these enhancements are due to image processing, not better cascade design, we have not considered them in our implementation. We would expect, however, to see similar computational gains in result of their application to the FCBoost cascade.

8. Conclusions

In this work we have addressed the problem of detector cascade learning by introducing the FCBoost algorithm. This algorithm optimizes a Lagrangian risk that accounts for both detector speed and accuracy with respect to a predictor that complies with the sequential decision making structure of the cascade architecture. By exploiting recursive properties of the latter, it was shown that many cascade predictors can be derived from generator functions, which are cascade predictors of two stages. Variants of FCBoost were derived for two members of this family, last-stage and multiplicative cascades, which were shown to generalize the popular independent and embedded stage cascade architectures. The concept of neutral predictors was exploited to integrate the search for cascade configuration into the boosting algorithm. In result, FCBoost can automatically determine 1) the number of cascade stages and 2) the number of weak learners per stage, by minimizing the Lagrangian risk. It was also shown that FCBoost generalizes adaboost, and is compatible with existing cost-sensitive extensions of boosting. Hence, it can be used to learn cascades of high detection rate. Experimental evaluation has shown that the resulting cascades outperform current state-of-the-art methods in both detection accuracy and speed.

Acknowledgments

This work was supported by NSF grant (NSF IIS-1208522) and the Technology Development Program for Commercializing System Semiconductor funded By the Ministry of Trade, industry & Energy(MOTIE, Korea), [No. 10041126, Title: International Collaborative R&BD Project for System Semiconductor].

References

P. Bartlett and M. Traskin. Adaboost is consistent. *Journal of Machine Learning Research*, 8:2347–2368, December 2007.

L. Bourdev and J. Brandt. Robust object detection via soft cascade. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 236–243, 2005.

S. Brubaker, M. Mullin, and J. Rehg. On the design of cascades of boosted ensembles for face detection. *International Journal of Computer Vision*, 77:65–86, 2008.

G. Carneiro, B. Georgescu, S. Good, and D. Comaniciu. Detection and measurement of fetal anatomies from ultrasound images using a constrained probabilistic boosting tree. *IEEE Transactions on Medical Imaging*, 27(9):1342 –1355, sept. 2008.

M. Collins, R. Schapire, and Y. Singer. Logistic regression, adaboost and Bregman distances. *Machine Learning*, 48(1-3):253–285, 2002.

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005.

P. Dollár, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *Proceedings of British Machine Vision Conference*, 2009.

P. Dollár, S. Belongie, and P. Perona. The fastest pedestrian detector in the west. In *Proceedings of British Machine Vision Conference*, 2010.

P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34 (4):743–761, 2012.

S. Du, N. Zheng, Q. You, Y. Wu, M. Yuan, and J. Wu. Rotated haar-like features for face detection with in-plane rotation. In *Proceedings of international conference on Interactive Technologies and Sociotechnical Systems*, pages 128–137, 2006.

J. Duchi and Y. Singer. Boosting with structural sparsity. *Proceedings of the International Conference on Machine Learning*, pages 297–304, 2009.

M. Dundar and J. Bi. Joint optimization of cascaded classifiers for computer aided detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Comp. and Sys. Science*, 1997.

J. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 1999.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28, 1998.

C. Lampert. An efficient divide-and-conquer cascade for nonlinear object detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1022 –1029, 2010.

C. Lampert, M. Blaschko, and T. Hofmann. Efficient subwindow search: A branch and bound framework for object localization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 2129–2142, 2009.

L. Lefakis and F. Fleuret. Joint cascade optimization using a product of boosted classifiers. In *Proceedings of the Neural Information Processing Systems Conference*, 2010.

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

R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *Proceedings of International Conference on Image Processing*, pages I–900 – I–903 vol.1, 2002.

C. Liu and H. Shum. Kullback-Leibler boosting. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 587–594, 2003.

H. Luo. Optimization design of cascaded classifiers. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 480–485, 2005.

H. Masnadi-Shirazi and N. Vasconcelos. High detection-rate cascades for real-time object detection. In *Proceedings of International Conference on Computer Vision*, volume 2, pages 1–6, 2007.

H. Masnadi-Shirazi and N. Vasconcelos. Cost-sensitive boosting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 99, 2010.

L. Mason, J. Baxter, P. Bartlett, and M. Frean. Functional gradient techniques for combining hypotheses. *Advances in Large Margin Classifiers*, pages 1221–246, 2000.

D. Mease and A. Wyner. Evidence contrary to the statistical view of boosting. *Journal of Machine Learning Research*, 9:131–156, 2008.

C. Messom and A. Barczak. Fast and efficient rotated haar-like features using rotated integral images. In *Proceedings of the Australasian Conference on Robotics and Automation*, 2006.

M. Pham and T. Cham. Fast training and selection of haar features using statistics in boosting-based face detection. In *Proceedings of IEEE International Conference on Computer Vision*, pages 1–7, 2007.

M. Pham, V. Hoang, and T. Cham. Detection with multi-exit asymmetric boosting. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1 – 8, 2008.

M. Pham, Y. Gao, V. Hoang, and T. Cham. Fast polygonal integration and its application in extending haar-like features to improve object detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 942 –949, 2010.

F. Porikli. Integral histogram: a fast way to extract histograms in cartesian spaces. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 829 – 836, 2005.

V. Raykar, B. Krishnapuram, and S. Yu. Designing efficient cascaded classifiers: Tradeoff between accuracy and cost. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 853–860, 2010.

D. Rumelhart, G. Hinton, and R. Williams. Example based learning for view-based human face detection. *Nature*, pages 533–536, 1968.

M. Saberian and N. Vasconcelos. Boosting classifer cascades. In *Proceedings of the Neural Information Processing Systems Conference*, 2010.

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

H. Schneiderman. Feature-centric evaluation for efficient cascaded object detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 29–36, 2004.

C. Shen, P. Wang, and H. Li. Lacboost and fisherboost: optimally building cascade classifiers. In *Proceedings of European Conference on Computer Vision*, pages 608–621, 2010.

C. Shen, S. Paisitkriangkrai, and J. Zhang. Efficiently learning a detection cascade with sparse eigenvectors. *IEEE Transactions on Image Processing*, 20(1):22 –35, jan. 2011.

J. Sochman and J. Matas. Waldboost - learning for time constrained sequential detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 150–157, 2005.

J. Sun, J. Rehg, and A. Bobick. Automatic cascade training with perturbation bias. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 276–283, 2004.

K. Kay Sung and T. Poggio. Example based learning for view-based human face detection. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 20:39–51, 1998.

O. Tuzel, F. Porikli, and P. Meer. Pedestrian detection via classification on riemannian manifolds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1713 –1727, 2008.

S. Vijayanarasimhan and K. Grauman. Efficient region search for object detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1401 –1408, 2011.

P. Viola and M. Jones. Robust real-time object detection. *Workshop on Statistical and Computational Theories of Vision*, 2001.

P. Viola and M. Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. In *Proceedings of the Neural Information Processing Systems Conference*, pages 1311–1318, 2002.

S. Walk, N. Majer, K. Schindler, and B. Schiele. New features and insights for pedestrian detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1030–1037, 2010.

X. Wang, T. Han, and S. Yan. An hog-lbp human detector with partial occlusion handling. In *Proceedings of IEEE International Conference on Computer Vision*, pages 32–39, 2009.

J. Wu, S. Brubaker, M. Mullin, and J. Rehg. Fast asymmetric learning for cascade face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3:369–382, 2008.

R. Xiao, L. Zhu, and H. Zhang. Boosting chain learning for object detection. In *Proceedings of International Conference on Computer Vision*, pages 709 –71, 2003.

R. Xiao, H. Zhu, H. Sun, and X. Tang. Dynamic cascades for face detection. In *Proceedings of International Conference on Computer Vision*, pages 1 – 8, 2007.

T. Zhang. Adaptive forward-backward greedy algorithm for learning sparse representations. *IEEE Transactions on Information Theory*, 57(7):4689 –4708, july 2011.

Q. Zhu, M. Yeh, K. Cheng, and S. Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1491 – 1498, 2006.