Learning Optimal Embedded Cascades

Mohammad Javad Saberian and Nuno Vasconcelos, *Senior Member*, *IEEE*

Abstract—The problem of automatic and optimal design of embedded object detector cascades is considered. Two main challenges are identified: optimization of the cascade configuration and optimization of individual cascade stages, so as to achieve the best tradeoff between classification accuracy and speed, under a detection rate constraint. Two novel boosting algorithms are proposed to address these problems. The first, RCBoost, formulates boosting as a constrained optimization problem which is solved with a barrier penalty method. The constraint is the target detection rate, which is met at all iterations of the boosting process. This enables the design of embedded cascades of known configuration without extensive cross validation or heuristics. The second, ECBoost, searches over cascade configurations to achieve the optimal tradeoff between classification risk and speed. The two algorithms are combined into an overall boosting procedure, RCECBoost, which optimizes both the cascade configuration and its stages under a detection rate constraint, in a fully automated manner. Extensive experiments in face, car, pedestrian, and panda detection show that the resulting detectors achieve an accuracy versus speed tradeoff superior to those of previous methods.

Index Terms—Computer vision, real-time object detection, embedded detector cascades, boosting.

1 INTRODUCTION

The problem of fast object detection has received substantial attention in computer vision since the introduction of a real-time face detector by Viola and Jones (VJ) in [1]. This detector is a cascade of simple to complex classifiers, designed with a combination of boosting and Haar wavelets which rejects most nonfaces with a few machine operations. Although the face detector has good performance, the learning algorithm is mostly a combination of heuristics, difficult to apply to other problems. One major difficulty is its reliance on two classes of parameters: *configuration parameters*, such as the numbers of cascade stages or weak learners per stage, and *rate parameters*, such as stage false positive and detection rates. Since cascade performance can vary nonintuitively with these parameters, their specification is far from trivial. This is compounded by difficulties such as an exponential increase of the miss rate on cascade length, or the need for example bootstrapping during learning. As a result, successful cascade training requires substantial experience in the design process, a massive example collection effort, and extensive trial-and-error.

Some of these problems have been addressed through various enhancements [2], [3], [4], [5], [6], [7]. A promising solution is the *embedded cascade* architecture, also known as *boosting chain* [8]. In this architecture, each stage differs from its predecessor by the addition of one or more weak learners [8], [9], [10], [11], [12], [13]. Since this divides the computation between stages very efficiently, and embedded cascades have good classification performance, this architecture underlies many recent cascade learning methods [9], [10], [11], [12], [13]. However, it is usually unclear how many weak learners should be added, per stage, to guarantee an optimal tradeoff between cascade speed and accuracy. Furthermore, embedded cascades are frequently learned with a two-step heuristic. A noncascaded classifier is first learned and then converted to an embedded cascade by introduction of intermediate exit points [11], [13]. Some postprocessing, such as application of a support vector machine (SVM) to the cascade outputs or threshold tuning, is also possible [8], [13]. In general, these steps cannot guarantee a cascade with the best tradeoff between detection speed and accuracy.

Most cascade learning algorithms are also unable to guarantee a specific detection rate. Since the cascade detection rate is strictly *smaller* than the individual rates of *all* its stages, a sensible value for the former requires the latter to be high. This forces each stage to operate in the saturation region of the receiver operating characteristic (ROC). Because in this region minimal variations of detection rate can produce large swings in false positive rate, it is critical that the individual rates are met tightly. This has been identified as a difficult problem since the early days of cascade design. In fact, Viola and Jones [1] could only address it with heuristics that require substantial manual guidance. Later solutions include cost-sensitive (CS) boosting [2], [11], [12], [14], [15], [16], [17], and optimal threshold adjustments [5], [10], [13]. While more principled, these have strong limitations of their own. In some cases, optimality requires conditions that do not hold for practical cascade design. In others, computationally intensive cross validation of learning parameters (e.g., classification cost factors) is required.

These problems are addressed by the two main contributions of this work. The first is a new boosting algorithm, RCBoost, that supports the *specification* of a detection rate and produces classifiers that meet this rate at *all* boosting iterations. This is done by formulating *boosting as a constrained optimization problem*, which combines the objective function of AdaBoost and a detection rate constraint. The optimization is solved with a barrier

• *The authors are with the Statistical Visual Computing Laboratory, University of California, San Diego, Room 5512, 9500 Gilman Drive, Mail code 0407, EBU 1, La Jolla, CA 92093-0407.
E-mail: {saberian, nvasconcelos}@ucsd.edu.*

*Manuscript received 5 July 2011; revised 15 Nov. 2011; accepted 12 Dec. 2011; published online 28 Dec. 2011.
Recommended for acceptance by H. Bischof.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMI-2011-07-0438.
Digital Object Identifier no. 10.1109/TPAMI.2011.281.*

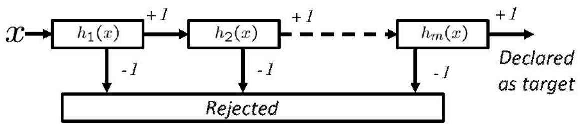

Fig. 1. The classifier cascade architecture.

method, assuring the feasibility of the solution at each gradient descent iteration. This guarantees a detector that *meets the target detection rate* without threshold adjustments or cost cross validation. The second is a procedure, ECBoost, that *searches the space of embedded cascades* for the detector of optimal tradeoff between classification error and speed. Optimality is defined by a Lagrangian that accounts for the two factors. Rather than designing a noncascaded classifier, the *embedded cascade is optimized directly*, using boosting-like gradient descent. In this way, in the process of searching for the cascade of optimal accuracy-speed tradeoff, ECBoost *automatically* determines the number of learners per cascade stage.

The two procedures are combined into a single boosting algorithm, RCECBoost, that *jointly* optimizes the cascade configuration and each of its stages while *guaranteeing* that a target cascade detection rate is met. The search for the cascade of optimal tradeoff between *false-positive rate and speed, at a given detection rate*, is performed in a single boosting run, with no need for parameter tuning or cross validation. RCECBoost is also shown to be fully compatible with standard bootstrapping procedures [1], [18], and produces state-of-the-art results on various object-detection tasks. The paper is organized as follows: Section 2 briefly reviews the problem of embedded cascade learning. RCBoost, ECBoost, and RCECBoost are then introduced in Sections 3, 4, and 5, respectively. Connections to previous work are discussed in Section 6, and an experimental evaluation is presented in Section 7. Finally, conclusions are drawn in Section 8.

2 Embedded Cascades

A binary classifier $h(x)$ maps an example $x \in \mathcal{X}$ to a class label $y \in \{-1, 1\}$. This is implemented as

$$h(x) = sign[f(x)], \tag{1}$$

where $f(x) : \mathcal{X} \rightarrow \mathbb{R}$ is a continuous-valued predictor defined over the example space $\mathcal{X}$. A classifier cascade $\mathcal{H}(x)$ implements a sequence of binary decisions

$$h_i(x) = sign[f_i(x)], \quad i = 1 \dots m, \tag{2}$$

as illustrated in Fig. 1. An example x is declared a target $(y = 1)$ if and only if it is declared a target by all stages, $(h_i(x) = 1, \forall i)$. Otherwise, it is rejected. The classifiers $h_i(x)$ are the *cascade stages*. They are usually implemented with weak learner ensembles, learned with boosting [19]. The *cardinality* of a cascade stage is the number of its weak learners. The *configuration* of a cascade is the vector of its stage cardinalities. The main advantage of this architecture is computational efficiency. If many examples are rejected by a few stages, the average classification time is very small.

An *embedded cascade*, or *boosting chain* [8], is a cascade whose predictor has the embedded structure

$$f_{i+1}(x) = f_i(x) + w_i(x). \tag{3}$$

$w_i(x)$ is the *predictor refinement* at stage i, consisting of a single [10], [11] or multiple weak learners [8], [12], [13]. In this way, each predictor refines its predecessors, and computation is shared by all stages. This enables cascade learning with a single boosting run. In fact, embedded cascades are usually learned by adding exit points to a noncascaded classifier. Single weak learner refinements lead to faster cascades and multiweak learner refinements to more accurate ones.

Optimal cascade design includes two main problems. The first is to determine the *optimal cascade configuration*. For a given detection problem, this is the configuration of best tradeoff between classification accuracy and speed. Given a measure of cascade performance that accounts for the two quantities, the search for the optimal cascade is a combinatorial problem since a classifier of m weak learners can be mapped into 2^{m-1} configurations. For realistic cascades with hundreds of weak learners, exhaustive search of all configurations is impossible.

Given the optimal cascade configuration, it remains to find the *optimal detector for each cascade stage*. A common assumption is that errors of different stages are independent:

$$D_{\mathcal{H}} = \prod_i D_i \leq \min_i\{D_i\}, \tag{4}$$

where $D_{\mathcal{H}}$ is the cascade detection rate and D_i that of stage i. It follows that $D_{\mathcal{H}}$ decays exponentially with cascade length m (e.g., if $D_i = 0.95, \forall i$, $D_{\mathcal{H}} = 0.95^m$). This usually implies that very high D_i are required to guarantee an acceptable $D_{\mathcal{H}}$, and *all* intermediate predictors $f_i(x)$ must guarantee *high* detection rates. This tends not to happen unless the cascade is learned under an *explicit* detection rate constraint.

3 Boosting with Rate Constraints

To address the second problem, we introduce a *rate constrained boosting* algorithm (RCBoost) which supports a detection rate constraint. We start by reviewing AdaBoost to recall the main boosting concepts.

3.1 AdaBoost

Boosting gained popularity with the introduction of AdaBoost [19], but has various interpretations. We adopt the view of [20], [21], where AdaBoost iterations are gradient descent steps with respect to the risk

$$R(f) = E_{X,Y}\{e^{-yf(x)}\}. \tag{5}$$

Let U be a set of weak learners. AdaBoost solves the optimization problem

$$\begin{cases} \min_{f(x)} & R(f) \\ s.t. & f(x) \in \mathcal{S}_u, \end{cases} \tag{6}$$

where $\mathcal{S}_u = Span(U)$ is the set of linear combinations of elements of U. Since information is only available through a training sample $S_t = \{(x_1, y_1), \dots, (x_n, y_n)\}$, the optimization is performed in the subspace U_n defined by the training points, e.g., projecting $f(x) \in \mathcal{S}_u$ into $[f(x_1), \dots, f(x_n)]^T \in \mathbb{R}^n$. R is then approximated by the empirical risk

$$R_e(f) \simeq \frac{1}{|S_t|} \sum_{x_i \in S_t} e^{-y_i f(x_i)}. \tag{7}$$

Starting with $f^0(x) = 0$, boosting updates follow the negative gradient at the current solution, $f^k(x)$:

$$-\nabla_{R_e(f^k)}(x_i) = -\left.\frac{\partial R_e(f^k(x) + \zeta I(x = x_i))}{\partial \zeta}\right|_{\zeta=0} \tag{8}$$

$$= \frac{-1}{|S_t|}\sum_{x_j \in S_t}\left.\frac{\partial e^{-y_j[f^k(x_j)+\zeta I(x_j=x_i)]}}{\partial \zeta}\right|_{\zeta=0} \tag{9}$$

$$= \frac{-1}{|S_t|}\frac{\partial}{\partial \zeta}\left.e^{-y_i[f^k(x_i)+\zeta]}\right|_{\zeta=0} \tag{10}$$

$$= \frac{y_i}{|S_t|}e^{-y_i f^k(x_i)} = \frac{y_i w_i^k}{|S_t|}, \tag{11}$$

where $I(x)$ is the indicator function

$$I(x) = \begin{cases} 1, & \text{if } x \text{ holds} \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

and

$$w_i^k = e^{-y_i f^k(x_i)}. \tag{13}$$

The negative gradient is projected into U_n, and the direction (weak learner) along which the projection has the largest magnitude,

$$g^*(x) = \arg\max_{g \in U_n}\langle g(x), -\nabla_{R_e(f^k)}(x)\rangle \tag{14}$$

$$= \arg\max_{g \in U_n}\frac{1}{|S_t|}\sum_i y_i w_i^k g(x_i), \tag{15}$$

is selected, where $<.,.>$ denotes the euclidean dot product. The optimal step size is then

$$\alpha^* = \arg\min_{\alpha}\ R_e(f^k + \alpha g^*). \tag{16}$$

If $g^*(x)$ is binary, i.e., $g^*(x) \in \{+1, -1\}$, then [19]

$$\alpha^* = \frac{1}{2}\log\frac{\sum_{i|y_i = g^*(x_i)} w_i^k}{\sum_{i|y_i \neq g^*(x_i)} w_i^k}. \tag{17}$$

These steps are summarized in Algorithm 1.

Algorithm 1. AdaBoost

```
**Input:** Training set $S_t = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, where
$y \in \{1, -1\}$ is the class label of example $\mathbf{x}$. Number
of weak learners in the final classifier $N$.
**Initialization:**
Set $k = 0$ and $f^k(x) = 0$.
**while** $k < N$ **do**
  Compute the weights $w_i^k = e^{-y_i f^k(x_i)}$.
  Select the best weak learner $g^*(x)$ with (15).
  Find the optimal step size $\alpha^*$ with (16).
  Update $f^{k+1}(x) = f^k(x) + \alpha^* g^*(x)$.
  $k = k + 1$
**end while**
**Output:** decision rule: $sign[f^N(x)]$
```

3.2 RCBoost

RCBoost is a boosting algorithm for the constrained optimization problem

$$\begin{cases} \min_{f(x)} & R_e(f) \\ s.t & R_D(f) \geq D_T, f(x) \in S_u, \end{cases} \tag{18}$$

where R_e is the risk of (7), D_T a target detection rate, and R_D the classifier's detection rate

$$R_D(f) = \int_{f(x)\geq 0} p(x|y=1)dx \tag{19}$$

$$= \int u[f(x)]p(x|y=1)dx \tag{20}$$

$$= E_{X|Y}\{u[f(x)]|y=1\}, \tag{21}$$

with $u(x) = I(x \geq 0)$ the Heaviside step. To guarantee a differentiable R_D, we use the popular approximation

$$u(x) \approx \hat{u}(x) = \frac{1 + tanh(\lambda x)}{2}, \tag{22}$$

where λ is a relaxation parameter. Combining this and the projection into U_n transforms (21) into

$$R_D(f) \simeq \frac{1}{|V^+|}\sum_{x_i \in V^+}\frac{1 + tanh(\lambda f(x_i))}{2}, \tag{23}$$

where V^+ is a set of positive examples. These could be the positive training examples or a validation set.

To meet the detection rate D_T after *each* boosting iteration, (18) requires a gradient descent algorithm that *guarantees* a feasible solution at each step. We adopt the family of *barrier methods* [22], which transform (18) into the unconstrained minimization of

$$J(\gamma, f) = R_e(f) + \gamma\mathbf{B}(R_D(f) - D_T). \tag{24}$$

The barrier $\mathbf{B}(R_D - D_T)$ assigns infinite penalty to constraint violations, forcing the solution to remain in the feasible set at *all* iterations. Gradient descent is repeated for a decreasing sequence γ_l, where the minimizer of $J(\gamma_{l-1}, f)$ is used to initialize the minimization of $J(\gamma_l, f)$. This guarantees continuous progress toward the solution of (18). In practice, the precise choice of γ_l is not critical; any positive decreasing sequence, convergent to zero, suffices.

We adopt a logarithmic barrier

$$\mathbf{B}(z) = \begin{cases} -\log z, & z > 0 \\ \infty, & z < 0, \end{cases} \tag{25}$$

leading to

$$J(\gamma, f) = \frac{1}{|S_t|}\sum_{x_j \in S_t} e^{-y_j f(x_j)} + \gamma\mathbf{B}\left(\sum_{x_j \in V^+}\frac{\hat{u}(f(x_j))}{|V^+|} - D_T\right), \tag{26}$$

where $\gamma > 0$. Given a feasible solution $f^k(x)$, the steepest descent direction for iteration $k+1$ is

$$-\nabla_{J(\gamma,f^k)}(x_i) = -\left.\frac{\partial J(\gamma, f^k(x) + \zeta I(x = x_i))}{\partial \zeta}\right|_{\zeta=0} \quad (27)$$

$$= \frac{-I(x_i \in S_t)}{|S_t|} \sum_{x_j \in S_t} \left.\frac{\partial}{\partial \zeta} e^{-y_j[f^k(x_j)+\zeta I(x_j=x_i)]}\right|_{\zeta=0} + \gamma I(x_i \in V^+), \quad (28)$$

$$\left.\frac{\partial \log(R_D[f^k(x_j) + \zeta I(x_j = x_i)] - D_T)}{\partial \zeta}\right|_{\zeta=0} = \frac{y_i w_i^k}{|S_t|} I(x_i \in S_t) + \gamma \frac{\xi_i^k}{|V^+|} I(x_i \in V^+), \quad (29)$$

with

$$w_i^k = e^{-y_i f^k(x_i)}, \quad (30)$$

$$\xi_i^k = \frac{\lambda}{2} \frac{1 - tanh^2(\lambda f^k(x_i))}{R_D(f^k) - D_T}. \quad (31)$$

The optimal weak learner is

$$g^*(x) = \arg\max_{g \in U_n} \langle g(x), -\nabla_{J(\gamma,f^k)}(x) \rangle \quad (32)$$

$$= \arg\max_{g \in U_n} \left\{ \sum_{x_i \in S_t} \frac{y_i w_i^k g(x_i)}{|S_t|} + \gamma \sum_{x_i \in V^+} \frac{\xi_i^k g(x_i)}{|V^+|} \right\} \quad (33)$$

and the optimal step size

$$\alpha^* = \arg\min_{\alpha} J(\gamma, f^k + \alpha g^*). \quad (34)$$

In general, there is no closed form for α^*, which is determined by a line search. Note that, by definition of barrier in (25), $J(\gamma, f)$ is infinite whenever the rate constraint is violated. Hence, the step α^* guarantees a feasible solution. The initial classifier is chosen to accept every example, $f^1(x) = \epsilon$, to guarantee a feasible starting point. As is common for barrier methods [22], γ is divided by 2 at every N_d iterations (e.g., $N_d = 5$). RCBoost is summarized in Algorithm 2.

Algorithm 2. RCBoost

```
**Input:** Training set $S_t$, validation set $V^+$, desired detection rate $D_T$, positive numbers $\epsilon, \lambda, \gamma$, Total number of weak learners in the classifier $N$ and $N_d$ number of iteration before halving $\gamma$.
**Initialization:**
Set $f^1(x) = \epsilon$, $k = 1$.
**while** $k < N$ **do**
  Compute the weights $w_i^k$ and $\xi_i^k$ with (30) and (31).
  Select the best weak learner $g^*(x)$ with (33).
  Find the optimal step size $\alpha^*$ with (34).
  Update $f^{k+1}(x) = f^k(x) + \alpha^* g^*(x)$.
  $k = k + 1$
  **if** $N_d \equiv 0 \ (mod\ N_d)$ **then**
    $\gamma = \gamma/2$
  **end if**
**end while**
**Output:** decision rule: $sign[f^N(x)]$
```

3.3 Properties

An analysis of $-\nabla_{J(\gamma,f^k)}(x_i)$ provides insight on the properties of RCBoost. For a point x_i in both S_t and V^+, (29) can be written as

$$-\nabla_{J(\gamma,f^k)}(x_i) = \frac{y_i}{|S_t|} w_i^k + \frac{\gamma}{|V^+|} \xi_i^k. \quad (35)$$

The first term is identical to the AdaBoost gradient of (11). It encourages classifiers of least error rate. As usual in boosting, it is small for points of large positive margin, i.e., correctly classified and far from the boundary. The second term encourages classifiers with the target detection rate. Note that $\hat{u}(.)$ is a smooth approximation to the Heaviside step, and its derivative a smooth approximation to the Dirac delta. Since ξ_i^k is the derivative of $\hat{u}(.)$ at x_i, at iteration k it is nonzero only for examples close to the boundary, increasing their impact on the gradient. This effect is modulated by the ratio $\gamma/[|V^+|(R_D(f^k) - D_T)]$. For small γ, this is a small quantity whenever R_D is larger than D_T. In this case, the second term is small, and the gradient is equivalent to that of AdaBoost. However, as R_D approaches D_T, the modulation increases and the second term enhances the influence of boundary points on the gradient. *This allows RCBoost to focus more on boundary points when there is pressure to violate the detection rate constraint*. Hence, the gradient step is steered away from the boundary, allowing the solution to stay within the feasible region. In summary, RCBoost is identical to AdaBoost when there is no pressure to violate the rate constraint, but can behave very differently as the constraint is approached. It can thus be seen as a generalization of AdaBoost, which inherits its interesting properties, e.g., simplicity and margin maximization, but *supports a rate constraint*. This justifies the name of *rate-constrained boosting*.

A second interesting property is that RCBoost can combine training and validation if V^+ is a validation set. In this case, the contribution of the training examples to the gradient is exactly the same as in AdaBoost, while examples in the validation set are used to enforce the rate constraint. Overall, the validation set provides a correction to AdaBoost, steering the optimization from constraint violations. A third interesting property is that RCBoost guarantees the target detection rate independently of the negative training examples. It thus automatically supports bootstrapping procedures that periodically replace easily classified negative examples with difficult ones [1], [18].

4 CASCADE CONFIGURATION

We next consider the problem of optimal cascade configurations. The decision rule implemented by a cascade, $\mathcal{H}$, of predictors $f_1, \ldots, f_m$ can be written as

$$\mathcal{H}(f_1, ..f_m)(x) = sign[\mathcal{C}(f_1, ..f_m)(x)], \quad (36)$$

where

$$\mathcal{C}(f_1, ..f_m)(x) = \begin{cases} f_j(x) & \text{if } f_j(x) < 0 \text{ and} \\ & f_i(x) \geq 0 \quad i = 1..j-1 \\ f_m(x) & \text{if } f_i(x) \geq 0 \quad i = 1..m-1 \end{cases} \quad (37)$$

is denoted the *cascade predictor*. Let

$$\mathcal{C}_m(x) \equiv \mathcal{C}(f_1, ..f_m)(x) \quad (38)$$

and note that, for an example x, either 1) the prediction $\mathcal{C}_m(x)$ is identical to that of last cascade stage, $f_m(x)$, or 2) x is rejected by the cascade composed of the previous stages, $\mathcal{C}_{m-1}$. This can be summarized as

$$\mathcal{C}_m = \mathcal{C}_{m-1}u[-\mathcal{C}_{m-1}] + u[\mathcal{C}_{m-1}]f_m, \tag{39}$$

where $u(.)$ is the Heaviside step, and we have omitted the dependence on x for notational simplicity. This recursion has two interesting properties.

Property 1. *The addition, to a cascade, of a stage identical to its last does not change its predictions:*

$$\mathcal{C}(f_1,\dots,f_m)(x) = \mathcal{C}(f_1,\dots,f_m,f_m)(x). \tag{40}$$

A proof of this property is given in Appendix A, which can be found in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2011.281.

Property 2. $\mathcal{C}(f_1,\dots,f_m)$ *is a linear function of the last stage predictor* f_m*:*

$$\mathcal{C}(f_1,\dots,f_m)(x) = \kappa(x) + \beta(x)f_m(x), \tag{41}$$

with coefficients

$$\kappa(x) = \mathcal{C}_{m-1}(x)u[-\mathcal{C}_{m-1}(x)], \tag{42}$$

$$\beta(x) = u[\mathcal{C}_{m-1}(x)]. \tag{43}$$

Proof. This property follows from (39) and the fact that κ, β only depend on $f_1 \dots f_{m-1}$ not f_m. □

Note that

$$\beta(x) = \prod_{j=1}^{m-1} u[f_j(x)], \tag{44}$$

since $u[\mathcal{C}_{m-1}(x)] > 0$ if and only if $f_j(x) \geq 0 \ \ \forall j < m$.

4.1 Cascade Risk Minimization

We next consider the minimization of the cascade risk:

$$R_e[\mathcal{C}(f_1,\dots,f_m)] = \frac{1}{|S_t|}\sum_{x_i\in S_t} e^{-y_i\mathcal{C}(f_1,\dots,f_m)(x_i)}. \tag{45}$$

As before, this is accomplished by gradient descent in U_n. A predictor $\mathcal{C}(f_1^k,\dots,f_m^k)$, with $m \leq k$, is available after the kth descent iteration. Two enhancements are possible at iteration $k+1$. The first is to augment the last stage f_m^k with a weak learner, i.e., to maintain the number of stages at m, make $f_j^{k+1} = f_j^k, \forall j < m$, and $f_m^{k+1} = f_m^k + g$. The second is to add a new stage f_{m+1}^{k+1}, i.e., make $f_j^{k+1} = f_j^k, \forall j \leq m$ and append a new f_{m+1}^{k+1} to the cascade.

4.1.1 Updating the Last Stage

We start by considering the best update under the first possibility. It follows from (41) that

$$\mathcal{C}\big(f_1^k,\dots,f_m^k+g\big)(x_i) = a_i^k + b_i^k g(x_i) \tag{46}$$

with

$$a_i^k = \kappa(x_i) + \beta(x_i)f_m^k(x_i), \tag{47}$$

$$b_i^k = \beta(x_i) = \prod_{j=1}^{m-1} u[f_j^k(x_i)]. \tag{48}$$

Note that, from (41) and (47),

$$a_i^k = \mathcal{C}\big(f_1^k,\dots,f_m^k\big)(x_i). \tag{49}$$

Given the solution $\mathcal{C}_m^k \equiv \mathcal{C}(f_1^k,\dots,f_m^k)$ at iteration k, the steepest descent update of the last stage is

$$-\nabla_{R_e(\mathcal{C}_m^k)}(x_i) = -\frac{\partial R_e\big[\mathcal{C}\big(f_1^k,\dots,f_m^k+\zeta I(x=x_i)\big)\big]}{\partial\zeta}\bigg|_{\zeta=0} \tag{50}$$

$$= -\frac{\partial R_e\big[a_i^k+\zeta b_i^k I(x=x_i))\big]}{\partial\zeta}\bigg|_{\zeta=0} \tag{51}$$

$$= \frac{y_i b_i^k}{|S_t|}e^{-y_i a_i^k}. \tag{52}$$

Hence, the weak learner selection rule is

$$g^* = \arg\max_{g\in U_n}\langle g(x), -\nabla_{R_e(\mathcal{C}_m^k)}(x)\rangle \tag{53}$$

$$= \arg\max_{g\in U_n}\frac{1}{|S_t|}\sum_{x_i\in S_t} y_i b_i^k w_i^k g(x_i) \tag{54}$$

with

$$w_i^k = e^{-y_i a_i^k} = e^{-y_i\mathcal{C}(f_1^k,\dots,f_m^k)(x_i)}. \tag{55}$$

Using (46) and (16), the optimal step size is

$$\alpha^* = \arg\min_\alpha R_e\big(\mathcal{C}\big(f_1^k,\dots,f_m^k+\alpha g^*\big)\big) \tag{56}$$

$$= \arg\min_\alpha \sum_{x_i\in S_t} e^{-y_i(a_i^k+\alpha b_i^k g^*(x_i))}, \tag{57}$$

where, for $g^*(x) \in \{+1,-1\}$,

$$\alpha^* = \frac{1}{2}\log\frac{\sum_{i|y_i=g^*(x_i)} b_i^k w_i^k}{\sum_{i|y_i\neq g^*(x_i)} b_i^k w_i^k}. \tag{58}$$

4.1.2 Adding a New Stage

We next consider the best update under the second possibility, where a new stage is added to the cascade. We start by considering a cascade in general form, i.e., whose stages are not embedded, with predictor $\mathcal{C}(f_1^k,\dots,f_m^k,g)(x)$. Note that consistency of gradient descent requires that taking no step, i.e., choosing $g(x) = 0$, leaves the predictor unaltered. This does not hold trivially since

$$\mathcal{C}\big(f_1^k,\dots,f_m^k,0\big)(x) \neq \mathcal{C}\big(f_1^k,\dots,f_m^k\big)(x). \tag{59}$$

To guarantee that the cascade output remains unaltered when $g(x) = 0$, it is sufficient to exploit (40) and define the new stage as

$$f_{m+1}^{k+1}(x) = f_m^k(x) + g(x). \tag{60}$$

This provides a mathematical justification for the embedded cascade structure: This structure is a sufficient condition for the learnability of detector cascades by gradient descent. Using (46),

$$\mathcal{C}(f_1^k, \dots, f_m^k, f_m^k + g)(x_i) = a_i^k + d_i^k g(x_i), \tag{61}$$

where

$$a_i^k = \mathcal{C}(f_1^k, ..., f_m^k, f_m^k)(x_i) \tag{62}$$

$$= \mathcal{C}(f_1^k, ..., f_m^k)(x_i), \tag{63}$$

$$d_i^k = \prod_{j=1}^{m} u\left[f_j^k(x_i)\right] \tag{64}$$

$$= b_i^k u\left[f_m^k(x_i)\right], \tag{65}$$

and (63), (65) follow from (40) and (48). The optimal weak learner and step size are derived as in the previous section, leading to

$$g^* = \arg\max_{g \in U_n} \frac{1}{|S_t|} \sum_{x_i \in S_t} y_i d_i^k w_i^k g(x_i), \tag{66}$$

$$\alpha^* = \arg\min_{\alpha} \sum_{x_i \in S_t} e^{-y_i(a_i^k + \alpha d_i^k g^*(x_i))}, \tag{67}$$

with the weights w_i^k of (55). For binary $g^*(x)$,

$$\alpha^* = \frac{1}{2} \log \frac{\sum_{i|y_i = g^*(x_i)} d_i^k w_i^k}{\sum_{i|y_i \neq g^*(x_i)} d_i^k w_i^k}. \tag{68}$$

4.2 ECBoost

From (45), (46), and (61), it follows that the update of the last cascade stage and the addition of a new cascade stage have similar risks. The only difference is the use of the *gating coefficients* b_i^k in (46) and d_i^k in (61). Note, from (64), that $d_i^k = 0$ if and only if x_i is rejected by *any* of the stages of $\mathcal{C}(f_1^k, ...f_m^k)$, i.e., if $\exists j \leq m$ such that $f_j^k(x_i) < 0$. Similarly, from (48), $b_i = 0$ if and only if $\exists j < m$ such that $f_j^k(x_i) < 0$. Hence, b_i^k and d_i^k are the same, up to the examples rejected by the mth cascade stage, for which $b_i^k > 0$ and $d_i^k = 0$. It follows that these examples influence the boosting process for the last stage update, but not for learning a new stage. Since detectors learned from larger pools of examples generalize better, the update of the last stage would always be the best choice for the minimization of (45). On the other hand, the elimination of examples is the mechanism by which cascaded detectors achieve fast classification.

To account for the two goals, we resort to a Lagrangian formulation, where the detector risk $R_e(\mathcal{C})$ of (45) is minimized under a complexity constraint. Complexity is measured by the number of machine operations, $T[\mathcal{C}(x)]$, required to classify example x using detector $\mathcal{C}$. This leads to the Lagrangian

$$\mathcal{L}(\mathcal{C}, \mu) = R_e(\mathcal{C}) + \mu \overline{T}(\mathcal{C}) \tag{69}$$

$$= \frac{1}{|S_t|} \sum_{x_i \in S_t} e^{-y_i \mathcal{C}(x_i)} + \frac{\mu}{|S_t^-|} \sum_{x_i \in S_t^-} T[\mathcal{C}(x_i)], \tag{70}$$

where S_t^- is the set of negative training examples and μ a Lagrange multiplier that controls the tradeoff between detection rate and speed. The restriction to S_t^- is mostly for compliance with the literature, where detector complexity is only evaluated for negative examples (which are overwhelmingly more frequent than positives, dominating detection complexity).

Since, given μ, the minimization of (69) guarantees the optimal tradeoff between classification risk and complexity, the search for the optimal cascade can be implemented with an extension to this cost of the gradient descent procedure above. This is the essence of *ECBoost,* which grows a cascade by computing its updates under the two strategies—1) adding a stage and 2) augmenting the last stage—and selecting the configuration for which (69) is smallest. At iteration k of ECBoost, the update (α_1^*, g_1^*) of the last stage, and the new stage (α_2^*, g_2^*) are computed with (54), (57) and (66), (67), respectively. The Lagrangian of (69) is then computed for the two cascades, and the one with the smallest cost selected. Note that, in this way, ECBoost can learn embedded cascades with a variable number of weak learners per stage. The coefficients $a^{k+1}, b^{k+1}, d^{k+1}$ can be computed with (49), (48), and (64), or recursively. In this case, when the last stage is updated:

$$f_j^{k+1} = f_j^k \quad j \leq m-1, \tag{71}$$

$$f_m^{k+1} = f_m^k + \alpha_1^* g_1^*, \tag{72}$$

$$a_i^{k+1} = a_i^k + b_i^k \alpha_1^* g_1^*(x_i), \tag{73}$$

$$b_i^{k+1} = b_i^k, \tag{74}$$

$$d_i^{k+1} = b_i^k u\left[f_m^{k+1}(x_i)\right], \tag{75}$$

while

$$f_j^{k+1} = f_j^k \quad j \leq m, \tag{76}$$

$$f_{m+1}^{k+1} = f_m^k + \alpha_2^* g_2^*, \tag{77}$$

$$a_i^{k+1} = a_i^k + d_i^k \alpha_2^* g_2^*(x_i), \tag{78}$$

$$b_i^{k+1} = d_i^k, \tag{79}$$

$$d_i^{k+1} = d_i^k u\left[f_{m+1}^{k+1}(x_i)\right] \tag{80}$$

holds if a new stage is added. The derivation of (73)-(80) is given in Appendix B, available in the online supplemental material.

4.3 Properties

ECBoost has various interesting properties. First, by comparing (54) and (66) with (15), it generalizes AdaBoost. In fact, the two algorithms only differ in the weights assigned to the training examples. In both (13) and (55), the weight w_i^k measures how well the training example x_i is classified at iteration k. As in AdaBoost, these weights discount well-classified examples, emphasizing the regions of $\mathcal{X}$ where the current predictor underperforms. The only difference is that, for ECBoost, the weights w_i^k are multiplied by the gating coefficients $b_i^k, d_i^k \in \{0,1\}$, which assign *zero weight* to regions of $\mathcal{X}$ rejected prior to the stage being updated or created. This is intuitive since examples in these regions will not reach the stage during cascade operation. It is also an advantage of ECBoost over the combination of noncascaded learning and rejection points, e.g., as in [11], [13]. Under the latter, the (noncascaded) learning of a weak learner is influenced by examples that it will never process once rejection points are inserted, and thus suboptimal. Hence, while ECBoost maintains the familiar emphasis of boosting on difficult examples (through w_i), the gating coefficients b_i and d_i prevent the *mismatch* between noncascaded training and cascaded operation that is characteristic of cascade design based on (a posteriori) threshold tuning.

A second significant property is that ECBoost *learns* the cascade configuration which minimizes an optimality criterion, the Lagrangian of (69), that accounts for both classification speed and accuracy. This leads to a fully *automated* cascade design process, which does not require prespecification of the configuration or any form of postprocessing. The inability to automate existing cascade design procedures, namely, that proposed by Viola and Jones [1], is a major bottleneck for the wide deployment of the cascade architecture.

A third interesting property is that ECBoost provides a mathematical justification for the bootstrapping procedure commonly used in the literature [1], [18]. This is a heuristic used to regenerate the training set after the design of each cascade stage. Examples rejected by the current cascade are eliminated from the training set of subsequent stages, and replaced by false positives collected from a validation set. Since the role of the gating coefficients of ECBoost is exactly to remove rejected examples (by assigning them zero weight for the subsequent design), ECBoost justifies the bootstrapping heuristic as a *necessary* step of the search for the cascade of best tradeoff between classification speed and risk. Note, however, that this only applies to the example removal component of bootstrapping. To replicate the addition of false positives, ECBoost must still be combined with the latter.

Finally, ECBoost is conceptually simple and can be implemented efficiently. While we have closely followed the derivation of AdaBoost, the procedures above can be easily adapted to any boosting method that has an interpretation as gradient descent, e.g., LogitBoost [20], RealBoost [23], or TangentBoost [24]. In fact, we next combine ECBoost with RCBoost.

5 Cascades with Rate Guarantees

ECBoost does not guarantee a detection rate for either the intermediate stages or the entire cascade. To overcome this limitation, we combine it with RCBoost, by replacing the Lagrangian of (69) with

$$\mathcal{L}(\mathcal{C},\mu,\gamma) = J(\gamma,\mathcal{C}) + \mu\overline{T}(\mathcal{C}), \tag{81}$$

where

$$J(\gamma,\mathcal{C}) = R_e(\mathcal{C}) + \gamma\mathbf{B}(R_D(\mathcal{C}) - D_T). \tag{82}$$

$\mathcal{C}$ is the embedded cascade predictor, γ a decreasing sequence, and $\mathbf{B}(z)$ the logarithmic barrier of (25). As in ECBoost, the best cascade update is computed by gradient descent with respect to $J(\gamma,\mathcal{C})$, under two strategies: adding a new stage versus updating the last. Equation (81) is then used to select the strategy of best tradeoff between false-positive rate and complexity.

Let $\mathcal{C}_m^k(x) = \mathcal{C}(f_1^k,\dots,f_m^k)(x)$ be the predictor after k gradient descent iterations, and a^k, b^k, d^k as in (48), (49), and (64). If $\mathcal{C}_m^k(x)$ is a feasible solution, the gradient for update of the last cascade stage is

$$-\nabla_{J(\gamma,\mathcal{C}_m^k)}(x_i) = -\left.\frac{\partial J\big(\gamma,\mathcal{C}\big[f_1^k(x_i)..f_m^k(x_i)+\zeta I(x=x_i)\big]\big)}{\partial\zeta}\right|_{\zeta=0} \tag{83}$$

$$= -\left.\frac{\partial J\big(\gamma, a_i^k + \zeta b_i^k I(x=x_i)\big)}{\partial\zeta}\right|_{\zeta=0} \tag{84}$$

$$\begin{aligned} &= -\left.\frac{\partial R_e\big(a_i^k+\zeta b_i^k I(x=x_i)\big)}{\partial\zeta}\right|_{\zeta=0} I(x_i\in S_t) \\ &+\gamma I(x_i\in V^+)\left.\frac{\partial \log\big(R_D\big(a_i^k+\zeta b_i^k I(x=x_i)\big)-D_T\big)}{\partial\zeta}\right|_{\zeta=0} \end{aligned} \tag{85}$$

$$= \frac{y_i b_i^k w_i^k}{|S_t|} I(x_i\in S_t) + \frac{\gamma b_i^k \xi_i^k}{|V^+|} I(x_i\in V^+), \tag{86}$$

where we have used (46) and

$$w_i^k = e^{-y_i\mathcal{C}_m^k(x_i)} = e^{-y_i a_i^k}, \tag{87}$$

$$\xi_i^k = \frac{\lambda}{2}\frac{1-tanh^2\big(\lambda\mathcal{C}_m^k(x_i)\big)}{R_D\big(\mathcal{C}_m^k\big)-D_T}. \tag{88}$$

The optimal weak learner is

$$g^* = \arg\max_{g\in U_n}\left\{\sum_{x_i\in S_t}\frac{y_i b_i^k w_i^k g(x_i)}{|S_t|} + \gamma\sum_{x_i\in V^+}\frac{b_i^k\xi_i^k g(x_i)}{|V^+|}\right\} \tag{89}$$

and the optimal step size

$$\alpha^* = \arg\min_\alpha J(\gamma, a^k + \alpha b^k g^*). \tag{90}$$

Equation (90) does not have a closed-form solution, and a line search is used to find α^*. Due to the infinite penalty assigned to constraint violations by (25), this step size guarantees a feasible solution.

The gradient for the addition of a new stage is

$$-\nabla_{J(\gamma,\mathcal{C}_{m+1}^k)}(x_i) = -\left.\frac{\partial J(\gamma, a_i^k + \zeta d_i^k I(x=x_i))}{\partial\zeta}\right|_{\zeta=0} \tag{91}$$

$$= \frac{y_i d_i^k w_i^k}{|S_t|} I(x_i\in S_t) + \frac{\gamma d_i^k \xi_i^k}{|V^+|} I(x_i\in V^+), \tag{92}$$

where w_i^k, ξ_i^k are given by (87), (88). The optimal weak learner is

$$g^* = \arg\max_{g \in U_n} \left\{ \sum_{x_i \in S_t} \frac{y_i d_i^k w_i^k g(x_i)}{|S_t|} + \gamma \sum_{x_i \in V^+} \frac{d_i^k \xi_i^k g(x_i)}{|V^+|} \right\} \quad (93)$$

and the optimal step size

$$\alpha^* = \arg\min_{\alpha} J\big(\gamma, a^k + \alpha d^k g^*\big), \quad (94)$$

found by a line search.

The two gradient steps are computed and the cascade configuration for which (81) is smallest is selected. Because the cascade has the embedded structure, all recursions previously derived for a^k, b^k, d^k still hold. This procedure is denoted *rate-constrained embedded-cascade boosting*, or RCECBoost for short, and summarized in Algorithm 3. Similarly to RCBoost, RCECBoost is initialized with $f(x) = \epsilon > 0$ so as to accept every example, and guarantees that the detection rate of the whole cascade is higher than D_T after each iteration, with no need for cross validation.

Algorithm 3. RCECBoost

```
  Input: Training set $S_t$, validation set $V^+$, desired detection rate $D$, tradeoff parameter $\mu$, number of weak learners $N$, barrier coefficient $\gamma$ and $N_d$ number of iteration before halving $\gamma$.
  Initialization:
  Set $f_1^1(x) = \epsilon$, $a_i^1 = \epsilon$, $b_i^1 = 1$, $d_i^1 = 1$, $m = 1$, $k = 1$.
  while $k < N$ do
    Compute $w_i^k$ and $\xi_i^k$ with (87) and (88).
    Find $(\alpha_1^*, g_1^*)$ for $\mathcal{C}' = \mathcal{C}(f_1^k, .., f_m^k + \alpha_1^* g_1^*)$, with $w_i^k, \xi_i^k, a_i^k, b_i^k$ and (89), (90).
    Find $(\alpha_2^*, g_2^*)$ for $\mathcal{C}'' = \mathcal{C}(f_1^k, .., f_m^k, f_m^k + \alpha_2^* g_2^*)$, with $w_i^k, \xi_i^k, a_i^k, d_i^k$ and (93), (94).
    if $\mathcal{L}(\mathcal{C}'', \mu, \gamma) < \mathcal{L}(\mathcal{C}', \mu, \gamma)$ then
      $f_{m+1}^{k+1} = f_m^k + \alpha_2^* g_2^*$
      Set $f_j^{k+1} = f_j^k \quad \forall j \leq m$.
      Increase number stages $m = m + 1$.
      Compute $a_i^{k+1}, b_i^{k+1}, d_i^{k+1}$ with (76)-(80).
    else
      $f_m^{k+1} = f_m^k + \alpha_1^* g_1^*$
      Set $f_j^{k+1} = f_j^k \quad \forall j < m$.
      Compute $a_i^{k+1}, b_i^{k+1}, d_i^{k+1}$ with (71)-(75).
    end if
    $k = k + 1$
    if $N_d \equiv 0 \ (mod\ N_d)$ then
      $\gamma = \gamma/2$
    end if
  end while
  Output: decision rule: $sign[\mathcal{C}(f_1^N(x), ..., f_m^N(x))]$
```

6 Relation to Previous Work

The embedded cascade learning procedures in the literature can be divided into three broad classes: a posteriori *threshold tuning*, *threshold optimization*, and *cost-sensitive boosting*. We next discuss the advantages of the proposed algorithms over these approaches.

6.1 Threshold Tuning

Threshold tuning methods start by designing a noncascaded detector. Thresholds are then introduced and tuned to produce a cascade with the target detection rate. A popular threshold tuning approach is the SoftCascade method [13]. A very large noncascaded classifier, $F(x) = \sum_i \alpha_i g_i(x)$, is first learned with AdaBoost and a modified bootstrap scheme [13]. This classifier is then converted to an embedded cascade by reordering weak learners, introducing exit points, and tuning the corresponding thresholds. This conversion can be seen as a search for the cascade configuration that solves the optimization problem

$$\begin{cases} \min\limits_{m, f_1, .. f_m} & R_{fp}(\mathcal{H}[f_1, \dots f_m]) \\ s.t & R_D(\mathcal{H}[f_1, \dots f_m]) = D_T \\ & \overline{T}(\mathcal{H}[f_1, \dots f_m]) = S_T, \end{cases} \quad (95)$$

where $R_{fp}(\mathcal{H})$ is the false positive rate of $\mathcal{H}$, and D_T and S_T are the target detection rate and complexity, respectively. When compared to our approach, these methods have two main problems.

The first is to compromise the generalization ability of boosting. While Friedman et al. [20] have shown that boosting learns the optimal predictor—the log-likelihood ratio (LLR) surface—for the binary classification problem at hand, this only holds asymptotically (infinite training samples). In the finite sample regime, the predictor is well approximated in a neighborhood, $\mathcal{N}(\mathcal{B})$, of the classification boundary $\mathcal{B}$ but poorly outside $\mathcal{N}(\mathcal{B})$ [25], [26]. This is not surprising since boosting's weighting mechanism, e.g., (13), concentrates the weight of the learning sample on $\mathcal{N}(\mathcal{B})$. However, it has the consequence that varying the threshold of a detector learned with AdaBoost is not equivalent to varying the threshold of the LLR. In general, there are no guarantees that a transformed classification boundary $\mathcal{B}'$ of detection rate D_T is the optimal boundary $\mathcal{B}_{D_T}$ (boundary of smallest false positive rate) at that rate. Hence, threshold tuning frequently produces suboptimal classifiers at the new detection rates [25], [26]. This is unlike the approach proposed in this work where boosting learns $\mathcal{B}_{D_T}$ directly. In this case, there is no loss of generalization.

A second problem is that, due to its combinatorial nature, threshold tuning requires suboptimal approximations. For example, because boosting is a gradient descent procedure in the space of weak learners, SoftCascade learning can be interpreted as

- perform gradient descent on the set U_n and store the sequence of gradient steps in a set $\mathcal{G} \ll U_n$,
- expand $\mathcal{G}$ into $\mathcal{G}^+$ by slightly perturbing each step, e.g., by varying weak learner thresholds,
- find the sequence of perturbed gradient steps in $\mathcal{G}^+$ that best solves (95).

In general, it is unlikely that this search over a limited subset of gradient steps can produce a solution superior to that of direct gradient descent under the constraints of (95), e.g., RCECBoost. The SoftCascade attempts to solve this problem by relying on a very large initial classifier (large $\mathcal{G}$). This substantially increases the learning complexity and does not necessarily provide better guarantees of optimality.

6.2 Threshold Optimization

A second class of methods optimizes thresholds *inside* boosting [7], [9], [10]. While the methods differ, a common thread is the use of classical decision theory to either 1) predict optimal thresholds or 2) formulate the entire problem. We will use WaldBoost [10], [27] as an example, due to its popularity and a very elegant connection to Wald's sequential decision theory [28]. It is a procedure for the solution of the optimization

$$\begin{cases} \min\limits_{m,f_1,..f_m} & \overline{T}(\mathcal{H}[f_1,\dots f_m]) \\ s.t & \forall i \leq m \quad R_D(f_i) \geq D_T \\ & \forall i \leq m \quad R_{fp}(f_i) \leq fp_T, \end{cases} \tag{96}$$

where D_T and fp_T are target detection and false positive rates, respectively. The optimal solution is a sequential rule that, at each step can accept, reject, or make no decision about the example to classify. Learning is essentially a combination of AdaBoost with two thresholds (that determine the reject/accept decisions) and bootstrapping. A stage is created per boosting iteration, its thresholds set according to Wald theory, and the training set bootstrapped.

While the connection to Wald's theory is elegant, the theory only applies to asymptotical sequential decision making. This is not the case of cascades, which have finitely many stages. Furthermore, the theory only applies to independent measurements, which are not available in cascade design. To address this and establish a connection to boosting, WaldBoost relies on Friedman et al. [20]. As discussed above, this only holds in the asymptotic regime, this time in terms of sample size. In summary, the theory behind WaldBoost only holds in the doubly asymptotical regime of infinite data and infinitely long cascades. It thus shares the limitations of the SoftCascade.

6.3 Cost-Sensitive Boosting

These problems are eliminated by cost-sensitive boosting methods, which learn the optimal boundary *directly*, through minimization of a risk that assigns a different cost to each type of error. For example, for any *optimal boundary* $\mathcal{B}_{D_T}$ of high detection rate D_T, there is a risk which assigns higher costs to misses than false positives so as to concentrate the sample weight on a neighborhood $\mathcal{N}(\mathcal{B}_{D_T})$. Rather than 1) learning the optimal boundary in a neighborhood $\mathcal{N}(\mathcal{B})$ and 2) transferring it by threshold adjustments, the *optimal* boundary $\mathcal{B}_{D_T}$ is learned directly.

Many cost-sensitive extensions of AdaBoost have been proposed. Some [14], [15], [16] are heuristic, simply adding cost-factors to its weight update rule. These algorithms are suboptimal, e.g., adopt suboptimal step sizes α, and underperform methods, such as Asymmetric Boosting [11] or AsymetricAdaBoost [2], derived from cost-sensitive extensions of the risk of (7) [11], [26]. These extensions are shown in Table 1, along with the resulting weight updates. The cost of each example is defined as

$$c(x) = C_+ I[y(x) = 1] + C_- I[y(x) = -1], \tag{97}$$

and the ratio $\frac{C_+}{C_-}$ determines the relative importance of positive and negative examples. For all methods,

$$-\nabla_{R_e(f^k)}(x_i) = y(x_i)w(x_i), \tag{98}$$

TABLE 1
Weight Updates and Risk Functions of Different Boosting Algorithms

Method	$w^{k+1}(x_i)$	Risk $R_e(f)$
AdaBoost [19]	$e^{-y_i f^k(x_i)}$	$\sum_i e^{-y_i f(x_i)}$
Asym-Ada [2]	$c_i e^{-y_i f^k(x_i)}$	$\sum_i c_i e^{-y_i f(x_i)}$
Asymmetric [11]	$c_i e^{-y_i c(x_i) f^k(x_i)}$	$\sum_i e^{-c_i y_i f(x_i)}$

y_i denotes the label of example x_i, $f^k(x)$ the predictor learned at the kth iteration, and c_i the cost factor of example x_i.

and implementation follows Algorithm 1, using the definitions of Table 1.

While addressing the limitations of threshold tuning, these methods have two substantial problems. First, while the costs (C_+, C_-) can be very intuitive for some problems, e.g., in fraud detection a false positive is known to cost C_+ and a miss C_- dollars, they are not available for cascade design, where only the detection rate D_T is specified. Albeit Neyman-Pearson's lemma guarantees that the optimal classifier for a given (C_+, C_-) is also optimal for some D_T, the mapping between (C_+, C_-) and D_T is usually unknown. In fact, this mapping varies from one problem to another. Hence, cost factors have to be found by cross validation. Second, even when cross validation is used and the overall classifier meets the target D_T, the same is not guaranteed for *each boosting iteration*. Hence, an embedded cascade created by adding exit points to a detector learned with CS-boosting has unpredictable detection rate. Both problems are compounded when the training set is modified, e.g., by the use of bootstrapping. In this case, a good set of cost factors before example replacement does not guarantee good performance after it. As a result, CS-boosting algorithms frequently misclassify too many positive examples to accommodate the new bootstrapped negatives.

6.4 RCECBoost

RCECBoost addresses all the problems above. The specification of μ in the Lagrangian of (81) is equivalent to that of an upper bound on complexity, $S_T(\mu)$, for the classification problem under consideration. This encodes the value of computational complexity to the cascade designer. Overall, the minimization of (81) is equivalent to solving the optimization problem

$$\begin{cases} \min\limits_{m,f_1,..f_m} & R_e(\mathcal{C}[f_1,\dots f_m]) \\ s.t & R_D(\mathcal{C}[f_1,\dots f_m]) \geq D_T \\ & \overline{T}(\mathcal{C}[f_1,\dots f_m]) \leq S_T(\mu), \end{cases} \tag{99}$$

where D_T is a target detection rate and $S_T(\mu)$ a target average complexity. Like the CS-boosting algorithms, it learns the optimal boundary *directly* in $\mathcal{N}(\mathcal{B}_{D_T})$. However, because this is done with the barrier penalty of (82), there is *no need for cross validation* of cost factors. The only parameters are the target detection rate D_T and the complexity constraint parameter μ. While the function $S_t(\mu)$ can vary across detection problems, our experiments (see Section 7) show that it is possible to learn detectors with a good compromise between speed and classification accuracy using a constant μ. In RCECBoost, the rate D_T is guaranteed for all boosting iterations. In fact, because the

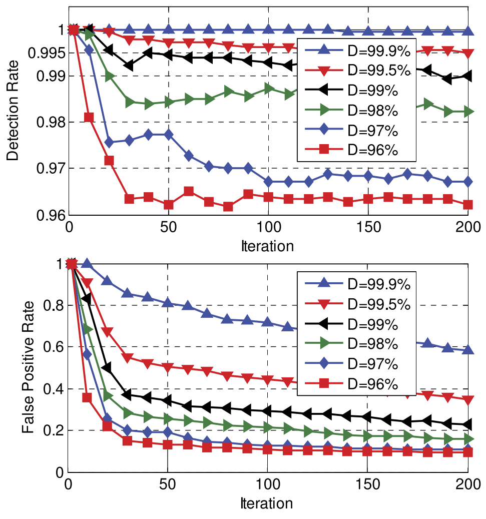

Fig. 2. Detection (top) and false positive rate (bottom) of RCBoost on the test set for target detection rates shown in the legend.

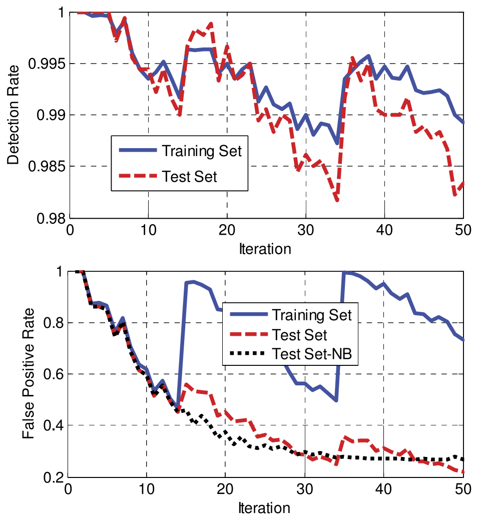

Fig. 3. Evolution of detection (top) and false positive (bottom) rates on the train and test set for RCBoost with bootstrapping. The test set false positive rate in the absence of bootstrapping is also shown (Test Set-NB) for comparison.

detection rate estimate of (23) only depends on a set V^+ of positive examples, RCECBoost even guarantees D_T independently of the negative training examples used. This is unlike cost sensitive boosting, for which the cost factors that guarantee D_T vary with the negative training samples.

7 Evaluation

In this section, we report on an extensive experimental evaluation of the algorithms proposed in this work. Four sets of experiments were conducted. The first addressed the properties of RCBoost as a general tool for detector design under detection rate constraints. Detector cascades were then considered in the second set. These experiments tested the ability of ECBoost to produce cascaded detectors with a good balance between classification risk and complexity. The third set addressed cascade learning under detection rate constraints, comparing RCECBoost to previous cascade learning methods. Finally, a fourth set compared RCECBoost cascades to detectors from the broader object detection literature. In all experiments, a pool of about 9,000 random images, provided by [17], was used for bootstrapping, and weak learners were thresholds on Haar wavelet features [1]. Since these features have nearly identical computation, $\overline{T}$ was defined in (69) and (81) as the average number of features evaluated, per example, by the classifier. For RCBoost and RCECBoost, the positive component of the training set was used as validation set V^+.

7.1 RCBoost

We start with two RCBoost experiments. Both addressed the problem of face detection, using a data set of 9,000 positive and 9,000 negative examples, of size 24×24. These were split into a training set of 7,200 positives and 7,200 negatives, and a test set containing the remaining examples. The first experiment tested the ability of RCBoost to maintain a detection rate uniformly higher than a target, across boosting iterations. For this, we trained detectors under the six target detection rates of Fig. 2. The figure presents plots of the detection and false positive rates as a function of the boosting iteration. Note that, even on the test set, the detection rates are quite close to the target. As expected, detectors with looser detection rate constraints achieve lower false-positive rates.

The second experiment tested the ability of RCBoost to maintain the target detection rate in the bootstrapping scenario. For this, we considered a variable training set, where all correctly classified negative examples were replaced by new false positives whenever the false positive rate dropped below 50 percent.[1] Fig. 3 presents the evolution of the detection and false positive rates, for $D_T = 98\%$. The sharp increases in training set false positive rate are aligned with the iterations where the training set was bootstrapped. On the test set, detection rate is always above target and false positive rate close to that obtained without bootstrapping, (Test Set-NB on the bottom plot).

7.2 ECBoost

The second set of experiments aimed to evaluate the performance of ECBoost. Besides face detection, they addressed the problems of car and pedestrian detection. The car data set was derived from the UIUC data [29]. In particular, we used its 550 positive examples (plus flipped replicas) as the positive set after resizing to 20×50 pixels. The negative examples were 12,000 random subwindows (of size 20×50) from the negative images of UIUC. The pedestrian data were based on the Caltech Pedestrian data set [30]. From the 11 sets of videos provided in this data set, we extracted, from sets 0-5, 9,714 positive and 10,000 negative examples. These were resized to 43×17 pixels. In all experiments of this section, these data sets were split fivefold and results averaged over five rounds. In each round, fourfolds were used for training and one for testing.

Since ECBoost does not provide detection rate guarantees, simply trading off classification speed for risk R_e, the detection rate of the resulting cascades quickly drops to unacceptably low values. This creates difficulty in the design of realistic cascades. The experiments in this set were thus mostly designed to understand the tradeoffs between

1. We usually adopt a threshold of 95 percent. Fifty percent was used in this experiment to magnify the variations, enabling easier visualization.

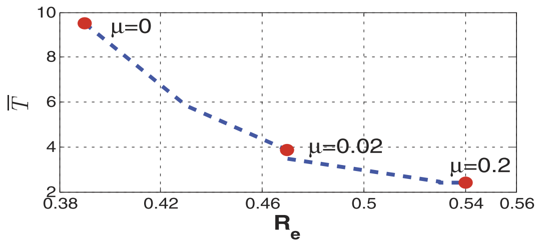

Fig. 4. Classification speed versus accuracy of ECBoost for different values of μ.

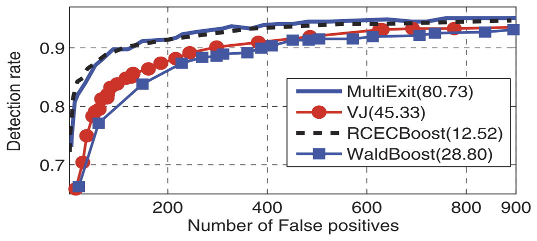

Fig. 5. ROC of various face detectors on MIT-CMU face data. The legend shows the average detection complexity $\overline{T}$ of each method.

detection accuracy and complexity. In particular, ECBoost was compared to 1) AdaBoost (equivalent to a version of ECBoost that always updates the last cascade stage), 2) ECBoost(1) (a version of ECBoost that always adds a new cascade stage), and 3) ChainBoost, where a detector is first learned with AdaBoost and an exit point inserted per weak learner. All detectors contained 24 weak learners.

We started by measuring the impact of the Lagrange multiplier μ of (69) on the performance of ECBoost cascades. Fig. 4 shows the classification risk, R_e, as a function of cascade complexity, $\overline{T}$, on the face data set, for cascades trained with different μ ($\mu = 0.2$ for leftmost point, $\mu = 0$ for rightmost). As expected, cascades learned with lower μ have lower error and higher complexity. We then set $\mu = 0.02$, and compared ECBoost with ChainBoost and AdaBoost, with the results of Table 2. As expected, AdaBoost had the lowest classification risks R_e, with the longest evaluation times. The cascade speedups ranged from 4.4 to 13.3 times, and most cascades were more than seven times faster than the AdaBoost detector. On the other hand, the cascade risk ranged from 1.2 to 2.4 times that of AdaBoost, and was below 1.7 times for most cascades. Overall, the cascades achieved a better tradeoff between speed and accuracy. This is reflected by their lower Lagrangian $\mathcal{L}$.

With regard to the performance of the various cascades, ECBoost(1) produced the fastest car and pedestrian detectors, while the face detector of ChainBoost was the fastest. In all cases, ECBoost learned the cascades of lowest Lagrangian $\mathcal{L}$. This is not surprising since it explicitly optimizes this quantity. It does, however, show that significant gains (26 percent over AdaBoost, 9.3 percent over ChainBoost, and 8.3 percent over ECBoost(1), on average) can be obtained by explicitly seeking the best tradeoff between speed and accuracy. Individually, a comparison between ECBoost(1) and ChainBoost reveals that accounting for the cascade structure during learning decreases $\mathcal{L}$ by about 1 percent, while a comparison between ChainBoost and ECBoost shows that an additional search for the cascade configuration of lowest $\mathcal{L}$ has a gain between 2 and 16 percent, depending on the data set. This gain is achieved by trading a moderate increase in complexity for a substantial decrease of the risk.

TABLE 2
Comparison of Cascade Learning Algorithms
($\mu = 0.02$, Detectors of 24 Weak Learners)

	Face			*Car*			*Pedestrian*		
Method	R_e	$\overline{T}$	$\mathcal{L}$	R_e	$\overline{T}$	$\mathcal{L}$	R_e	$\overline{T}$	$\mathcal{L}$
AdaBoost	**0.31**	24	0.79	**0.24**	24	0.72	**0.61**	24	1.09
ChainBoost	0.55	**2.4**	0.60	0.59	1.97	0.63	0.82	3.86	0.89
ECBoost(1)	0.54	2.41	0.59	0.59	**1.81**	0.62	0.83	**3.21**	0.89
ECBoost	0.47	3.85	**0.54**	0.42	5.34	**0.53**	0.76	5.41	**0.87**

7.3 RCECBoost

The performance of RCECBoost was compared to a number of algorithms in the literature. WaldBoost [10] was chosen to represent threshold optimization methods, and the multiexit cascade method of [12] to represent CS-boosting methods. To the best of our knowledge, this is the method that achieves the current best results in standardized data sets. Since it has been previously shown to outperform threshold tuning methods such as SoftCascade [12], these were not implemented. For completeness, the comparison also included the method of Viola and Jones (VJ) [1]. In all experiments, WaldBoost, multiexit, and VJ were bootstrapped when a new stage was added to the cascade. For RCECBoost, we used $\mu = 0.02$, and bootstrapping whenever the false positive rate dropped to 95 percent. For VJ and multiexit cascades, we used 20 stages, each with a target false positive rate of 50 percent and a detection rate $D_T^{\frac{1}{20}}$, respectively. For WaldBoost, following [10], we set $B = 0$ and $A = \frac{1}{1-D_T}$.

Face detection. Since state-of-the-art face detectors are based on the cascade architecture and face detection is the standard benchmark for cascaded detectors, we start with this task. For all methods, we trained a face detector with 99,638 Haar features and $D_T = 95\%$. The VJ and Multiexit cascades had 20 stages, while WaldBoost learned an embedded detector with 1,000 stages, each containing one new weak learner. RCECBoost produced an embedded detector with about 640 stages, 57 percent of which had one, 24 percent two, 9 percent three, and 10 percent more new weak learners. Fig. 5 presents the resulting ROCs on the MIT-CMU face data set. The legend also shows the average detection complexity $\overline{T}$ of each method. The RCECBoost cascade is more accurate than those of VJ and WaldBoost, at 3.6 and 2.3 times faster, respectively. With respect to multiexit, it has similar detection performance but is about 6.5 times faster. Overall, RCECBoost has the clear best performance.

Pedestrian detection. We next considered pedestrian detection. Note that the combination of cascades and Haar wavelets is not necessarily the best solution for this task [30], where edge-like feature such as HOG [31] can obtain better performance.[2] Nevertheless, the pedestrian task can

2. According to [32], the best current pedestrian detection results are due to [33]. This approach combines a cascaded detector with a fast multiscale method to compute Haar-like features over multiple channels, including gray scale, gradients, and color. The gradient information is an approximation to the HOG descriptor.

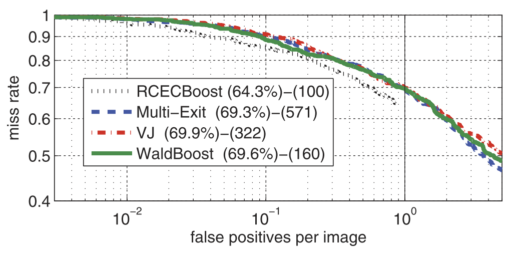

Fig. 6. Miss versus false positive rates for pedestrian detection. The legend shows the miss rate at one false positive per image (first value) and the average detection complexity $\overline{T}$ (second value).

be used to compare detector cascades. For this, cascades were learned with $D_T = 98\%$ and 114,771 Haar features. Detection performance was evaluated on a test set, disjoint from the training set, containing 1 out of every 30 frames of videos in sets 0-5 of Caltech [30], using the software provide by its authors. Again, the VJ and Multiexit cascades had 20 stages, and WaldBoost learned an embedded cascade of about 1,800 stages, each containing a single new weak learner. RCECBoost produced an embedded detector of about 1,000 stages, 56 percent of which had one, 19 percent two, 10 percent three, and 15 percent more new weak learners. Fig. 6 shows the miss versus false positive rate of all detectors, as computed by the software provided with the data set, in the near scale regime [30]. For each method, the legend shows the miss rate at one false positive per image (first value) and the average detection complexity $\overline{T}$ (second value). The RCECBoost cascade has the lowest miss rate and is the fastest. The closest performance is that of WaldBoost, with 7 percent less accuracy and 50 percent larger detection time. VJ, multiexit, and WaldBoost have similar accuracy, but the multiexit cascade is again substantially slower than all others (5× slower than RCECBoost).

7.4 Comparison to Other Methods

In this section, we compare the performance of different methods and architectures on two object detection problems. The comparison is based on 1) classification accuracy and 2) computational complexity. Complexity is measured as the average time (in seconds) elapsed per detection.[3]

Car detection. We start with some experiments on car detection using the UIUC single-scale and multiscale car (side view) data sets [29]. The single-scale data set contains 170 images with 200 cars of roughly equal size (100×40). The multiscale data set contains 108 images with 139 cars of multiple sizes. These data sets are interesting because results from a large number of methods are available for them. For example, Leibe et al. [34] proposed a combination of an implicit shape model (ISM) and minimum description length, Lampert et al. [35] the combination of an SVM, hierarchical spatial pyramid kernels, and an efficient subwindow search (ESS), while Fritz et al. [36] integrated an SVM, ISM, and local kernels, Fergus et al. [37] proposed part-based models learned by expectation maximization, and Mutch and Lowe [38] the combination of an SVM and a biologically inspired HMAX network. In the realm of cascaded detectors, Schneiderman [39] proposed a cascade with histogram weak learners, while Wu and Nevatia [40] used edgelet features in a RealBoost cascade.

3. The times reported in this section are either those reported in the original papers, or the result of running the algorithms on a dual core 2.6 GHz CPU.

TABLE 3
Comparison of Car Detectors on the UIUC Data Set

	Single Scale		*Multi Scale*	
Method	EER	Time(s)	EER	Time(s)
Agarwal [29]	77.5%	2.5	40.6%	12
ESS(4 × 4) [35]	98.5%	> 100	92.1%	> 100
ESS(10 × 10) [35]	98.5%	> 1000	98.6%	> 1000
Fergus [37]	88.5%	$10 - 15$	-	-
Leibe [34]	97.5%	$2 - 3$	95%	$4 - 5$
Fritz [36]	88.6%	N.R	87.8%	N.R
Mutch [38]	99.96%	> 3	90.6%	> 3
Schneiderman [39]	97%	< 1	-	-
Wu [40]	97.5%	0.200	93.5%	0.600
RCECBoost	99%	0.040	92.1%	0.229

A car detector was trained with RCECBoost, using $\mu = 0.02, D_T = 98\%$ and 179,213 Haar features. Table 3 presents the detection rate at equal error rate (EER) and average processing time required, per image, for all methods. The detection and false positive rates are computed as in [29], where a true detection is declared if its center is inside a ground-truth ellipse.

Several important observations can be made from the table. First, recent methods achieve very high accuracy on the both the single and multiscale data sets. Since differences in performance of 1 percent correspond to the detection of a few examples, there is a tendency to declare the data sets as "solved." We note, however, that this is not the case when a processing time constraint holds. In fact, among the existing methods, the only remotely close to real-time implementations are those based on detector cascades [39], [40]. These methods have relatively low performance. On the other hand, the methods of highest accuracy tend to have very large processing times, e.g., several minutes for [35], when 10 pyramid levels are used.

Second, RCECBoost cascade is orders of magnitude faster than most other detectors. Its processing times range from 40 to 230 milliseconds, i.e., about 2-5 times faster than the next best speeds, which are obtained by the cascades of [39], [40]. These speedups are complemented by a gain in detection accuracy of 1.5 percent on the single-scale data set, and a loss of the same magnitude for the multiscale data, i.e., an overall equivalent detection accuracy. Third, no method does very well on the multiscale data when complexity is taken into account. While the detection performance of the cascades, 92-93 percent, is substantially inferior to the best results, 98.6 percent for ESS (10×10), the complexity of the most accurate methods is unacceptable for most applications of practical interest. For example, ESS (10×10) is about 5,000 times slower than the RCECBoost cascade.

The fact that computation complexity should not be disregarded is well illustrated by the performance of ESS (4×4), which has the same detection rate as the RCECBoost cascade but is 500 times slower. Hence, while ESS (10×10) could be considered a "better" detector than the RCECBoost cascade, ESS (4×4) is definitely not. Overall, the RCECBoost cascade achieves the best compromise between

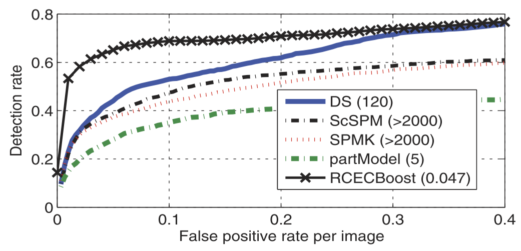

Fig. 7. Detection rate versus FPPI for panda detection. The numbers in the legend are the average detection times (seconds) per image.

detection accuracy and complexity. We would also argue that the car data set should not be declared "solved" as there is plenty of room for improvement when complexity is accounted.

Panda detection. While the car data set is one of the most mature in object detection, we finish with a very recent data set, which explicitly tests the main weaknesses of currently popular detection architectures [41]. This is a data set of a wildlife exhibit, a panda habitat, of much larger size (2,518 training and 2,500 test images of size 240×320), and wide variability of object scale, pose, background, and occlusion. The panda examples were rescaled to size 27×31, and RCECBoost used to learn a cascade with 128,274 Haar wavelets, $\mu = 0.02$, and $D_T = 98\%$. Detection performance was evaluated as in [41]. Fig. 7 presents the curves of detection rate versus number of false positives per image (FPPI) produced by a number of methods, including a discriminant saliency model (DS) proposed in [41], the discriminatively trained part-based model (part Model) popular in the PASCAL literature [42], the sparse coded spatial pyramid matching (ScSPM) method of [43], and the spatial pyramid matching kernel (SPMK) method of [44], which are state-of-the-art (single descriptor) methods in the Caltech101 and 15 scenes benchmarks. The numbers in the legend are the average detection times (seconds) per image.

Other than RCECBoost, these curves were reported in [41]. In this data set, the RCECBoost cascade achieves the best performance even when complexity is *not* taken into account. This is particularly true at low FPPI, e.g., while the previous best reported detection rate for a FPPI of 0.1 was 50 percent [41], the RCECBoost cascade achieves a detection rate of about 70 percent. With regard to detection speed, RCECBoost requires about 47 milliseconds to scan each image, which is suitable for real-time detection. This is between 200 and 4,000 times faster than the other methods!

8 Conclusion

The challenges of embedded cascade design are rooted in the limited ability of current boosting algorithms to 1) maintain a detection rate throughout learning and 2) search for the optimal cascade configuration. In this work, we have addressed these problems with two new boosting algorithms: RCBoost, which provides detection rate guarantees throughout the learning process, and ECBoost, which searches for the cascade configuration with optimal tradeoff between classification accuracy and speed. The two algorithms were then combined into a single procedure, RCECBoost, that optimizes the cascade configuration under a detection rate constraint, in a fully automated manner. Experimental evaluation on face, car, pedestrian, and panda detection has shown that the resulting cascades achieve a substantially better speed/accuracy tradeoff than previous approaches.

References

[1] P. Viola and M.J. Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* 2001.

[2] P. Viola and M. Jones, "Fast and Robust Classification Using Asymmetric Adaboost and a Detector Cascade," *Proc. Advances in Neural Information and Processing System,* 2001.

[3] X. Hou, C.-L. Liu, and T. Tan, "Learning Boosted Asymmetric Classifiers for Object Detection," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* 2006.

[4] S. Li and Z. Zhang, "Floatboost Learning and Statistical Face Detection," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 26, no. 9, pp. 1112-1123, Sept. 2004.

[5] H. Luo, "Optimization Design of Cascaded Classifiers," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* vol. 1, pp. 480-485, 2005.

[6] C. Liu and H.-Y. Shum, "Kullback-Leibler Boosting," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* vol. 1, pp. 587-594, 2003.

[7] S.C. Brubaker, J. Wu, J. Sun, M.D. Mullin, and J.M. Rehg, "On the Design of Cascades of Boosted Ensembles for Face Detection," *Int'l J. Computer Vision,* vol. 77, pp. 65-86, 2008.

[8] R. Xiao, L. Zhu, and H.-J. Zhang, "Boosting Chain Learning for Object Detection," *Proc. IEEE Int'l Conf. Computer Vision,* pp. 709-715, 2003.

[9] R. Xiao, H. Zhu, H. Sun, and X. Tang, "Dynamic Cascades for Face Detection," *Proc. IEEE Int'l Conf. Computer Vision,* pp. 1-8, 2007.

[10] J. Sochman and J. Matas, "Waldboost-Learning for Time Constrained Sequential Detection," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* pp. 150-156, 2005.

[11] H. Masnadi-Shirazi and N. Vasconcelos, "High Detection-Rate Cascades for Real-Time Object Detection," *Proc. IEEE Int'l Conf. Computer Vision,* 2007.

[12] M.-T. Pham, V.-D. Hoang, and T.-J. Cham, "Detection with Multi-Exit Asymmetric Boosting," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 1-8, 2008.

[13] L. Bourdev and J. Brandt, "Robust Object Detection via Soft Cascade," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* pp. 236-243, 2005,

[14] W. Fan, S.J. Stolfo, J. Zhang, and P.K. Chan, "Adacost: Misclassification Cost-Sensitive Boosting," *Proc. Int'l Conf. Machine Learning,* 1999.

[15] K.M. Ting, "A Comparative Study of Cost-Sensitive Boosting Algorithms," *Proc. Int'l Conf. Machine Learning,* pp. 983-990, 2000.

[16] A. Wong, Y. Sun, and Y. Wang, "Parameter Inference of Cost-Sensitive Boosting Algorithms," *Proc. Int'l Conf. Machine Learning and Data Mining in Pattern Recognition,* 2005.

[17] J. Wu, S.C. Brubaker, M.D. Mullin, and J.M. Rehg, "Fast Asymmetric Learning for Cascade Face Detection," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 30, no. 3, pp. 369-382, Mar. 2008.

[18] K.-K. Sung and T. Poggio, "Example-Based Learning for View-Based Human Face Detection," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 20, no. 1, pp. 39-51, Jan. 1998.

[19] Y. Freund and R.E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *Proc. European Conf. Computational Learning Theory,* 1995.

[20] J. Friedman, T. Hastie, and R. Tibshirani, "Additive Logistic Regression: A Statistical View of Boosting," *Annals of Statistics,* vol. 28, pp. 337-407, 2000.

[21] L. Mason, J. Baxter, P. Bartlett, and M. Frean, "Boosting Algorithms as Gradient Descent," *Proc. Advances in Neural Information Processing Systems,* 2000.

[22] J. Nocedal and S. Wright, *Numerical Optimization.* Springer Verlag, 1999.

[23] R.E. Schapire and Y. Singer, "Improved Boosting Algorithms Using Confidence-Rated Predictions," *Machine Learning,* vol. 37, pp. 297-336, 1999.

[24] H. Masnadi-Shirazi, V. Mahadevan, and N. Vasconcelos, "On the Design of Robust Classifiers for Computer Vision," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2010.

[25] D. Mease and A. Wyner, "Evidence Contrary to the Statistical View of Boosting," *J. Machine Learning Research,* vol. 9, pp. 131-156, 2008.

[26] H. Masnadi-Shirazi and N. Vasconcelos, "Cost-Sensitive Boosting," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 33, no. 2, pp. 294-309, Feb. 2011.

[27] J. Sochman, "Learning for Sequential Classification," PhD dissertation, Czech Technical Univ., 2009.

[28] A. Wald, *Sequential Analysis.* Dover, 1947.

[29] S. Agarwal, A. Awan, and D. Roth, "Learning to Detect Objects in Images via a Sparse, Part-Based Representation," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 26, no. 11, pp. 1475-1490, Nov. 2004.

[30] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian Detection: A Benchmark," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2009.

[31] S. Maji, A.C. Berg, and J. Malik, "Classification Using Intersection Kernel Support Vector Machines Is Efficient," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 1-8, 2008.

[32] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian Detection: An Evaluation of the State of the Art," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 34, no. 4, pp. 743-761, Apr. 2012.

[33] P. Dollár, S. Belongie, and P. Perona, "The Fastest Pedestrian Detector in the West," *Proc. British Machine Vision Conf.,* 2010.

[34] B. Leibe, A. Leonardis, and B. Schiele, "Robust Object Detection with Interleaved Categorization and Segmentation," *Int'l J. Computer Vision,* vol. 77, pp. 259-289, 2008.

[35] C. Lampert, M. Blaschko, and T. Hofmann, "Efficient Subwindow Search: A Branch and Bound Framework for Object Localization," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 31, no. 12, pp. 2129-2142, Dec. 2009.

[36] M. Fritz, B. Leibe, B. Caputo, and B. Schiele, "Integrating Representative and Discriminant Models for Object Category Detection," *Proc. IEEE Int'l Conf. Computer Vision,* vol. 2, pp. 1363-1370, 2005.

[37] R. Fergus, P. Perona, and A. Zisserman, "Object Class Recognition by Unsupervised Scale-Invariant Learning," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* vol. 2, pp. 264-271, 2003.

[38] J. Mutch and D. Lowe, "Multiclass Object Recognition with Sparse, Localized Features," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* vol. 1, pp. 11-18, 2006.

[39] H. Schneiderman, "Feature-Centric Evaluation for Efficient Cascaded Object Detection," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* vol. 2, pp. 29-36, 2004.

[40] B. Wu and R. Nevatia, "Simultaneous Object Detection and Segmentation by Boosting Local Shape Feature Based Classifier," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 1-8, 2007.

[41] S. Han and N. Vasconcelos, "Biologically Plausible Detection of Amorphous Objects in the Wild," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition Workshop,* 2011.

[42] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan, "Object Detection with Discriminatively Trained Part-Based Models," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 32, no. 9, pp. 1627-1645, Sept. 2010.

[43] J. Yang, K. Yu, Y. Gong, and T. Huang, "Linear Spatial Pyramid Matching Using Sparse Coding for Image Classification," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 1794-1801, 2009.

[44] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* vol. 2, pp. 2169-2178, 2006.

Mohammad Javad Saberian received the BS degrees in electrical engineering and computer science from Sharif University of Technology, Iran, in 2008. He is currently working toward the PhD degree at the University of California, San Diego, in the Electrical and Computer Engineering Department in the Statistical Visual Computing Laboratory. He was the recipient of a UC San Diego fellowship in 2008 and Yahoo Key Scientific Challenges award in 2011. His research interests are in machine learning and computer vision.

Nuno Vasconcelos received the licenciatura in electrical engineering and computer science from the Universidade do Porto, Portugal, in 1988, and the MS and PhD degrees from the Massachusetts Institute of Technology in 1993 and 2000, respectively. From 2000 to 2002, he was a member of the research staff at the Compaq Cambridge Research Laboratory, which in 2002 became the HP Cambridge Research Laboratory. In 2003, he joined the Electrical and Computer Engineering Department at the University of California, San Diego, where he heads the Statistical Visual Computing Laboratory. He is the recipient of a US National Science Foundation (NSF) CAREER award, a Hellman Fellowship, and has authored more than 100 peer-reviewed publications. His work spans various areas, including computer vision, machine learning, signal processing and compression, and multimedia systems. He is a senior member of the IEEE.