Branch-and-Bound Search for Training Cascades of Classifiers[*]

Dariusz Sychel$^{1[0000-0001-9835-869X]}$, Przemysław Klęsk$^{1[0000-0002-5579-187X]}$, and Aneta Bera$^{1[0000-0002-0456-9451]}$

Faculty of Computer Science and Information Technology, West Pomeranian University of Technology, ul. Żołnierska 49, 71-210 Szczecin, Poland
{dsychel,pklesk,abera}@wi.zut.edu.pl

Abstract. We propose a general algorithm that treats cascade training as a tree search process working according to the *branch-and-bound* technique. The algorithm allows to reduce the *expected number of features* used by an operating cascade — a key quantity we focus on in the paper. While searching, we observe suitable lower bounds on partial expectations and prune tree branches that cannot improve the best-so-far result. Both exact and approximate variants of the approach are formulated. Experiments pertain to cascades trained to be face or letter detectors with Haar-like features or Zernike moments being the input information, respectively. Results confirm shorter operating times of cascades obtained owing to the reduction in the number of extracted features.

Keywords: Cascade of classifiers · Branch-and-bound tree search · Expected number of features.

1 Introduction

Branch-and-bound technique is a useful tool in computer science. Multiple application examples can be named— let us mention DNA regulatory motif finding [8] and α-β pruning in games, just to give two examples from quite remote fields. In this paper we adopt the technique to train cascades of classifiers.

Cascades were in principle designed to work as classifying systems operating under the following two conditions: (1) very large number of incoming requests, (2) significant classes imbalance. The second condition should not be seen as a difficulty but rather a favorable setting that makes the whole idea viable. Namely, a cascade should vary its computational efforts depending on the contents of an object to be classified. Objects that are obvious negatives (non-targets) should be recognized fast, using only a few features extracted. Targets, or objects resembling, them are allowed to employ more features and time for computations.

Despite the development of deep learning, recent literature shows that cascades of classifiers are still widely applied in detection systems or batch classification jobs. Let us list a few examples: crowd analysis and people counting [1],

[*] This work was financed by the National Science Centre, Poland. Research project no.: 2016/21/B/ST6/01495.

human detection in thermal images [6], localization of white blood cells [4], eye tracking [11,9], detection of birds near high power electric lines [7].

There exist a certain *average* value of the computational cost incurred by an operating cascade. It can be mathematically defined as an *expected value* and, in fact, calculated explicitly for a given cascade (we do this in Section 2.3) in terms of: the number of features applied on successive stages, false alarm and detection rates on successive stages, probability distribution from which the data is drawn. Since the true distribution underlying the data is typically unknown in practice, the exact expected value cannot be determined. Interestingly though, it can be accurately approximated using just the feature counts and false alarm rates.

Training procedures for cascades are time-consuming, taking days or even weeks. As Viola and Jones noted in their pionieering work [18], cascade training is a difficult combinatorial optimization involving many parameters: number of stages, number of features on successive stages, selection of those features, and finally decision thresholds. The problem has not been ultimately solved yet. Viola and Jones tackled it by imposing the final requirements the whole cascade should meet in order to be accepted, defined by a pair of numbers (A, D), where A denotes the largest allowed false alarm rate (FAR), and D the smallest allowed detection rate (sensitivity). Due to probabilistic properties of cascade structure, one can translate final requirements onto per-stage requirements as geometric means: $a_{\max} = A^{1/K}$ and $d_{\min} = D^{1/K}$, where K is the fixed number of stages.

Many modifications to cascade training have been introduced over the years. Most of them try out different: feature selection approaches, subsampling methods, or are simply tailored to a particular type of features [3,12,10,17] (e.g. Haar, HOG, LBP, etc.). Some authors obtain modified cascades by designing new boosting algorithms that underlie the training [15,14], but due to mathematical difficulties, the expected number of features is seldom the main optimization criterion. One of few exceptions is an elegant work by Saberian and Vasconcelos [14]. The authors use gradient descent to optimize explicitly a Lagrangian representing the trade-off between cascade's error rate and the operating cost (expected value). They use a trick that translates non-differentiable recursive formulas to smooth ones using hyperbolic tangent approximations. The approach is analytically tractable but expensive, because all cascade stages are kept open while training. In every step one has to check *all* variational derivatives based on features at disposal for *all* open stages.

The main contribution of this paper is an algorithm — or in fact a general framework — for training cascades of classifiers via a tree search approach and the *branch-and-bound* technique. Successive tree levels correspond to successive cascade stages. Sibling nodes represent variants of the same stage with different number of features applied. We provide suitable formulas for lower bounds on the expected value that we optimize. During an ongoing search, we observe the lower bounds, and whenever a bound for some tree branch is greater than (or equal to) the best-so-far expectation, the branch becomes pruned. Once the search is finished, one of the paths from the root to some terminal node indicates the cascade with the smallest expected number of features. Apart from the exact approach to pruning, we additionally propose an approximate one, using suitable predictions of expected values.

2 Preliminaries

2.1 Notation

Throughout this paper we use the following notation:

- K — number of cascade stages,
- $n = (n_1, n_2, \ldots, n_K)$ — numbers of features used on successive stages,
- $(a_1, a_2, \ldots, a_K)$ — FAR values on successive stages (false alarm rates),
- $(d_1, d_2, \ldots, d_K)$ — sensitivities on successive stages (detection rates),
- A — required FAR for the whole cascade,
- D — required detection rate (sensitivity) for the whole cascade,
- $F = (F_1, F_2, \ldots, F_K)$ — ensemble classifiers on successive stages (the cascade),
- A_k — FAR observed up to k-th stage of cascade ($A_k = \prod_{1\leqslant i\leqslant k} a_i$),
- D_k — sensitivity observed up to k-th stage of cascade ($D_k = \prod_{1\leqslant i\leqslant k} d_i$),
- $(p, 1-p)$ — true probability distribution of classes (unknown in practice),
- $\mathcal{D}, \mathcal{V}$ — training and validation data sets,
- $\#$ — set size operator (cardinality of a set),
- $\|$ — concatenation operator (to concatenate cascade stages).

The probabilistic meaning of relevant quantities is as follows. The final requirements (A, D) demand that: $P\left(F(\mathbf{x})=+\,|y=-\right) \leqslant A$ and $P\left(F(\mathbf{x})=+\,|y=+\right) \geqslant D$, whereas false alarm and detection rates observed on particular stages are, respectively, equal to:

$$\begin{aligned} a_k &= P\left(F_k(\mathbf{x})=+\,|y=-, F_1(\mathbf{x})=\cdots=F_{k-1}(\mathbf{x})=+\right), \\ d_k &= P\left(F_k(\mathbf{x})=+\,|y=+, F_1(\mathbf{x})=\cdots=F_{k-1}(\mathbf{x})=+\right). \end{aligned} \tag{1}$$

2.2 Classical cascade training algorithm (Viola-Jones style)

The classical cascade training algorithm given below (Algorithm 1) can be treated as a reference for new algorithms we propose.

Please note, in the final line of the pseudocode, that we return $(F_1, F_2, \ldots, F_k)$ rather than $(F_1, F_2, \ldots, F_K)$. This is because the training procedure can potentially stop earlier, when $k < K$, provided that the final requirements (A, D) for the entire cascade are already satisfied i.e. $A_k \leqslant A$ and $D_k \geqslant D$.

The step "Adjust decision threshold" requires a more detailed explanation. The real-valued response of any stage can be suitably thresholded to obtain either some wanted sensitivity or FAR. Hence, the resulting $\{-1, +1\}$-decision of a stage is, in fact, calculated as the sign of expression

$$F_k(\mathbf{x}) - \theta_k,$$

where θ_k represents the decision threshold. Suppose $(v_1, v_2, \ldots, v_{\#\mathcal{P}})$ denotes a sequence of sorted, $v_i \leqslant v_{i+1}$, real-valued responses of a new cascade stage F_{k+1} obtained on positive examples (subset $\mathcal{P}$). Then, the $d_{\min}$ per-stage requirement can be satisfied by simply choosing: $\theta_{k+1} = v_{\lfloor (1-d_{\min})\cdot\#\mathcal{P}\rfloor}$.

Algorithm 1 VJ-style training algorithm for cascade of classifiers

```
procedure TrainVJCascade($\mathcal{D}$, $A$, $D$, $K$, $\mathcal{V}$)
    From $\mathcal{D}$ take subsets $\mathcal{P}$, $\mathcal{N}$ with positive and negative examples, respectively.
    $F := ()$                                    ▷ initial cascade — empty sequence
    $a_{\max} := A^{1/K}$, $d_{\min} := D^{1/K}$, $A_0 := 1$, $D_0 := 1$, $k := 0$.
    while $A_k > A$ do
        $n_{k+1} := 0$, $F_{k+1} := 0$, $A_{k+1} := A_k$, $a_{k+1} := A_{k+1}/A_k$.
        while $a_{k+1} > a_{\max}$ do
            $n_{k+1} := n_{k+1} + 1$.
            Train new weak classifier $f$ using $\mathcal{P}$ and $\mathcal{N}$
            $F_{k+1} := F_{k+1} + f$.
            Adjust decision threshold $\theta_{k+1}$ for $F_{k+1}$ to satisfy $d_{\min}$ requirement.
            Use cascade $F \| F_{k+1}$ on validation set $\mathcal{V}$ to measure $A_{k+1}$ and $D_{k+1}$.
            $a_{k+1} := A_{k+1}/A_k$.
        $F := F \| F_{k+1}$.
        if $A_{k+1} > A$ then
            $\mathcal{N} := \emptyset$.
            Use cascade $F$ to populate set $\mathcal{N}$ with false detections
                sampled from non-target images.
        $k := k + 1$
    return $F = (F_1, F_2, \ldots, F_k)$.
```

2.3 Expected number of extracted features

Definition-based formula A cascade stops operating after a certain number of stages. It does not stop in the middle of a stage. Therefore the possible outcomes of the random variable of interest, describing the disjoint events, are: n_1, n_1+n_2, $\ldots$, $n_1+n_2+\cdots+n_K$. Hence, by the definition of expected value, the expected number of features can be calculated as follows:

$$E(n) = \sum_{1\leqslant k\leqslant K} \Big(\sum_{1\leqslant i\leqslant k} n_i\Big) \left(p\Big(\prod_{1\leqslant i<k} d_i\Big)(1-d_k)^{[k<K]} + (1-p)\Big(\prod_{1\leqslant i<k} a_i\Big)(1-a_k)^{[k<K]} \right), \tag{2}$$

where $[\cdot]$ is an indicator function.

Incremental formula and its approximation By grouping the terms in (2) with respect to n_k the following alternative formula can be derived:

$$E(n) = \sum_{1\leqslant k\leqslant K} n_k \left(p \prod_{1\leqslant i<k} d_i + (1-p) \prod_{1\leqslant i<k} a_i \right). \tag{3}$$

Obviously, in practical applications the true probability distribution underlying the data is unknown. Since the probability p of the positive class is very small (typically $p < 10^{-4}$), the expected value can be accurately approximated using only the summands related to the negative class as follows:

$$\widehat{E}(n) = \sum_{1 \leqslant k \leqslant K} n_k \prod_{1 \leqslant i < k} a_i \approx E(n). \tag{4}$$

It is also interesting to remark that in the original Viola and Jones' paper [18] the authors proposed an incorrect formula to estimate the expected number of features, namely:

$$E_{\mathrm{VJ}}(n) = \sum_{k=1}^{K} n_k \prod_{i=1}^{k-1} r_i, \tag{5}$$

where r_i represents the "positive rate" of i-th stage. This is equivalent to

$$E_{\mathrm{VJ}}(n) = \sum_{k=1}^{K} n_k \prod_{i=1}^{k-1} (pd_i + (1-p)a_i). \tag{6}$$

Please note that by multiplying positive rates of stages, one obtains mixed terms of form $d_i \cdot a_j$ that do not have any probabilistic sense. For example for $k = 3$ the product under summation becomes $(pd_1 + (1-p)a_1)\,(pd_2 + (1-p)a_2)$, with the terms $d_1 a_2$ and $a_1 d_2$ having no sense, because a fixed data point does not change its class label while traveling along the cascade.

3 Cascade training as a tree search

In stage-wise training procedures, each stage, once fixed, must not be altered. The paper [14], discussed in the introduction, represents an opposite approach, where all stages can be extended with a weak classifier at any time. The approach we propose is in-between the two mentioned above. It provides more flexibility than stage-wise training and simultaneously avoids high complexity of [14].

We treat cascade training as a tree search process. The root of the tree represents an empty cascade. Successive tree levels correspond to successive cascade stages. Each non-terminal tree node is going to have an odd number of children nodes. They will represent variants of a subsequent stage with slightly different number of features. The children will be processed recursively from left to right until the stop condition is met. It should be understood that the nodes are not simply generated mechanically but, in fact, trained as ensemble classifiers.

The size of the tree shall be controlled by two integer parameters L and C, predefined by the user. To keep the tree fairly small, the branching of variants shall take place only at L top-most levels, e.g. $L = 2$. At those levels the branching factor will be equal to C, an odd number, e.g. $C = 5$. At deeper levels the branching factor will be one. Therefore, the actual branching shall affect only initial stages having the largest impact on the expected number of features. Once the tree search is finished, one of the paths from the root to some terminal node shall indicate the best cascade i.e. having the smallest expectation.

For notation purposes, children nodes being variants of the same stage use an additional subindex. For example, the classifier $F_{1,0}$ denotes the main variant of

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50423-6_2

the first stage (using a certain number of features) and is graphically represented as the *middle* child. Its *left* siblings $F_{1,-1}, F_{1,-2}, \ldots$ denote classifiers using fewer features (one less, two less, etc.). The *right* siblings $F_{1,+1}, F_{1,+2}, \ldots$ use more features than the middle child (one more, two more, etc.). This notation will be used only locally within single resursive calls (due to global ambiguity).

3.1 Pruning search tree using current partial expectations — exact branch-and-bound

During an ongoing tree search (combined with cascade training) one can observe *partial* values for the expected value of interest — formula (4). Suppose a new $(k+1)$-th stage has been completed, revealing n_{k+1} features. The formula

$$\widehat{E}\Big((n_1,\ldots,n_{k+1})\Big) = \sum_{1\leqslant j\leqslant k} n_j \prod_{1\leqslant i<j} a_i + n_{k+1} \prod_{1\leqslant i<k+1} a_i = \widehat{E}\Big((n_1,\ldots,n_k)\Big) + n_{k+1} \prod_{1\leqslant i<k+1} a_i. \tag{7}$$

expresses the partial expectation for the extended cascade in an incremental manner. It should be clear that whenever a partial expectation for some tree branch is greater than (or equal to) the best-so-far exact expectation, say $\widehat{E}\big((n_1,\ldots,n_{k+1})\big) \geqslant \widehat{E}^*$, then there is no point in pursuing that branch further down the tree. In other words, pruning can be applied because formula (7) provides a lower bound on the final unknown expectation.

Fig. 1 provides a symbolic illustration of a search tree with pruning. In the figure, the subindexes $E_1, E_2, \ldots$ are meant to indicate chronologically the partial expected values observed on the successive branches as the tree is being traversed from left to right. Crossed-out lines represent the pruned branches.

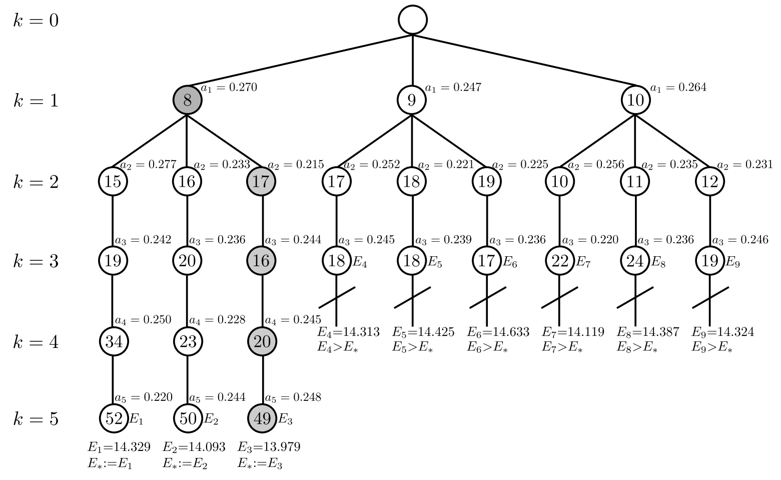

Fig. 1. Cascade training as a tree search with pruning — example illustration.

Algorithm 2 represents our proposition stated as a recursion. A single recursive call can be summarized as follows. It takes as input a partial cascade F with k stages and trains the new $(k+1)$-th stage in its main variant $F_{k+1,0}$. We refer to it as the *middle child*. Then, the algorithm "branches" the stage (if level not greater than L) by creating clones of the middle child with fewer features: $F_{k+1,-1}, F_{k+1,-2}, \ldots$ (*left children*), and with more features: $F_{k+1,+1}, F_{k+1,+2}, \ldots$ (*right children*). The algorithm iterates over all children and performs recursive calls to train their subsequent stages provided that the lower bound (7) on the final expectation is not worse than the best expectation $\widehat{E}^*$ so far. A recursion path, representing some cascade, reaches its stopping point when final requirements (A, D) are satisfied and when its expected value is strictly less than $\widehat{E}^*$ (initially, set to ∞). The outermost recursion call is

$$\textsc{TrainTreeCascade}\,(\mathcal{D}, A, D, K, 0, \mathcal{V}, (), L, C, \text{null}, \infty)$$

yielding a pair of results: the best cascade F^* and its expectation $\widehat{E}^*$.

Inside the subroutine TrainStage we train a single ensemble using per-stage requirements. They can be calculated a standard geometric means (classical VJ-style), leading to constant per-stage requirements for the whole training, or as updated geometric means (UGM): uniform or greedy. The formulas below represent the three options.

$$\text{VJ:}\quad a_{\max,k+1} = A^{1/K}, \qquad d_{\min,k+1} = D^{1/K}. \tag{8}$$

$$\text{UGM:}\quad a_{\max,k+1} = \Big(A\Big/\prod_{1\leqslant i\leqslant k} a_i\Big)^{1/(K-k)}, \qquad d_{\min,k+1} = \Big(D\Big/\prod_{1\leqslant i\leqslant k} d_i\Big)^{1/(K-k)}. \tag{9}$$

$$\text{UGM-G:}\quad a_{\max,k+1} = A^{(k+1)/K}\Big/\prod_{1\leqslant i\leqslant k} a_i, \qquad d_{\min,k+1} = D^{(k+1)/K}\Big/\prod_{1\leqslant i\leqslant k} d_i. \tag{10}$$

3.2 Pruning search tree using expectation predictions — approximate branch-and-bound

Suppose we have completed the training of stage $k+1$ and would like to make a prediction about the partial expectation for stage $k+2$ without training it. Obviously, the training of any stage is time-consuming, hence a significant gain would be benefited by not wasting time on a stage that is not going to improve the best-so-far expectation. Observe that when the stage $k+1$ is completed, we get to know two new pieces of information: n_{k+1} and a_{k+1}. That second piece is not needed to calculate formula (7) for stage $k+1$, but it is needed for stage $k+2$. Therefore, the only unkown preventing us from calculating the exact partial expectation for stage $k+2$ is n_{k+2}. We are going to approximate it.

As cascade experiments on real-data show, the counts of features $(n_k)_{k=1,\ldots,K}$ typically form a non-decreasing sequence. There exist counter-examples, but in the vast majority of cases it is true that $n_{k+1} \geqslant n_k$. Therefore, to build our

Algorithm 2 Training cascade of classifiers via tree search with exact pruning

```
procedure TrainTreeCascade($\mathcal{D}$, $A$, $D$, $K$, $k$, $\mathcal{V}$, $F$, $C$, $L$, $F^*$, $\widehat{E}^*$)
    From $\mathcal{D}$ take subset $\mathcal{P}$ with all positive examples, and subset $\mathcal{N}$ with all negative examples.
    Train stage for middle child: $F_{k+1,0}$ :=TrainStage($\mathcal{P}$, $\mathcal{N}$, $K$, $k$, $\mathcal{V}$, $F$).
    Use cascade $F\|F_{k+1,0}$ on validation set $\mathcal{V}$ to measure $A_{k+1,0}$ and $D_{k+1,0}$.
    if $k > L$ then
        $C := 1$.
    for $c := -1, -2, \ldots, -\lfloor C/2 \rfloor$ do                    ▷ left children
        Create $F_{k+1,c}$ by cloning $F_{k+1,c+1}$.
        Remove last weak classifier from $F_{k+1,c}$.
        Adjust decision threshold $\theta_{k+1,c}$ for $F_{k+1,c}$ to satisfy $d_{\min,k+1}$ requirement.
        Use cascade $F\|F_{k+1,c}$ on validation set $\mathcal{V}$ to measure $A_{k+1,c}$ and $D_{k+1,c}$.
    for $c := 1, 2, \ldots, \lfloor C/2 \rfloor$ do                       ▷ right children
        Create $F_{k+1,c}$ by cloning $F_{k+1,c-1}$.
        Train new weak classifier $f$ using $\mathcal{P}$ and $\mathcal{N}$
        $F_{k+1,c} := F_{k+1,c} + f$.
        Adjust decision threshold $\theta_{k+1,c}$ for $F_{k+1,c}$ to satisfy $d_{\min,k+1}$ requirement.
        Use cascade $F\|F_{k+1,c}$ on validation set $\mathcal{V}$ to measure $A_{k+1,c}$ and $D_{k+1,c}$.
    for $c := -\lfloor C/2 \rfloor, \ldots, 0, \ldots, \lfloor C/2 \rfloor$ do           ▷ all children
        Calculate expectation $\widehat{E}$ for cascade $F\|F_{k+1,c}$ using (7).
        if $A_{k+1,c} > A$ and $\widehat{E} < \widehat{E}^*$ then
            Prepare new training set $\mathcal{D}_{k+1,c}$ and new validation set $\mathcal{V}_{k+1,c}$.
            $(F^*, \widehat{E}^*)$:=TrainTreeCascade($\mathcal{D}_{k+1,c}$, $A$, $D$, $K$, $k$+1, $\mathcal{V}_{k+1,c}$, $F\|F_{k+1,c}$,
                                        $L$, $C$, $E^*$, $F^*$)
        else if $A_{k+1,c} \leqslant A$ and $\widehat{E} < \widehat{E}^*$ then
            $\widehat{E}^* := \widehat{E}$,   $F^* := F\|F_{k+1,c}$.
            return $(F^*, \widehat{E}^*)$.
    return $(F^*, E^*)$.

procedure TrainStage($\mathcal{P}$, $\mathcal{N}$, $K$, $k$, $\mathcal{V}$, $F$)
    $n_{k+1} := 0$, $F_{k+1} := 0$, $A_{k+1} := A_k$.
    Calculate per-stage requirements
    $(a_{\max,k+1}, d_{\min,k+1})$ using (8) or (9) or (10).
    $a_{k+1} := A_{k+1}/A_k$.
    while $a_{k+1} > a_{\max,k+1}$ do
        $n_{k+1} := n_{k+1} + 1$.
        Train new weak classifier $f$ using $\mathcal{P}$ and $\mathcal{N}$.
        $F_{k+1} := F_{k+1} + f$.
        Adjust decision threshold $\theta_{k+1}$ for $F_{k+1}$
        to satisfy $d_{\min,k+1}$ requirement.
        Use cascade $F\|F_{k+1}$ on validation
        set $\mathcal{V}$ to measure $A_{k+1}$ and $D_{k+1}$.
        $a_{k+1} := A_{k+1}/A_k$.
    return $F_k$.
```

prediction it could potentially be sufficient to lowerbound n_{k+2} by n_{k+1}. Instead, we prefer to propose a safer parameterized approach — by assuming:

$$n_{k+2} \geqslant \alpha\, n_{k+1}, \tag{11}$$

where parameter α could be selected e.g. from $[0.5, 1.5]$ interval. The following lines demonstrate explicitly the prediction we are going to apply:

$$\begin{aligned}\widehat{E}\Big((n_1,\dots,n_{k+2})\Big) &= \widehat{E}\Big((n_1,\dots,n_k)\Big) + n_{k+1}\prod_{1\leqslant i<k+1} a_i + n_{k+2}\prod_{1\leqslant i<k+2} a_i\\ &\approx \widehat{E}\Big((n_1,\dots,n_k)\Big) + n_{k+1}\prod_{1\leqslant i<k+1} a_i + \alpha\, n_{k+1}\Big(\prod_{1\leqslant i<k+1} a_i\Big)a_{k+1} \equiv \widehat{E}_\alpha. \end{aligned}\tag{12}$$

The influence of parameter α can be described as follows. By lowering α, one decreases the risk of pruning a branch incorrectly, but simultaneously one strengthens the underestimation of the expected value, which can lead to training continuation despite a negligible chance of improvment. In contrast, higher α values lead to more pruning but with some risk of missing the optimum solution. Additionally, it is worth to remark that the prediction we make is only for *one* stage ahead, ignoring all subsequent stages. Since those stages shall too contribute their summands to the final expectation then this suggests that high α values should still be safe, especially for initial levels.

Algorithm 3 represents the described approach for cascade training based on tree search and approximate pruning.

Algorithm 3 Training cascade of classifiers algorithm via tree search with approximate pruning

```
procedure TrainTreeCascadeApprox($\mathcal{D}$, $A$, $D$, $K$, $k$, $\mathcal{V}$, $F$, $C$, $L$, $F^*$, $\widehat{E}^*$, $\alpha$)
    ⋮                                                    ▷ initial steps same as in TrainTreeCascade

    for $c := -\lfloor C/2\rfloor, \dots, 0, \dots, \lfloor C/2\rfloor$ do                    ▷ all children
        if $A_{k+1,c} > A$ then
            Calculate expectation prediction $\widehat{E}_\alpha$ for $F\|F_{k+1,c}\|F_{k+2,\cdot}$ using (12).
            if $\widehat{E}_\alpha < \widehat{E}^*$ then
                Prepare new training set $\mathcal{D}_{k+1,c}$ and new validation set $\mathcal{V}_{k+1,c}$.
                $(F^*, \widehat{E}^*)$:=TrainTreeCascadeApprox($\mathcal{D}_{k+1,c}$, $A$, $D$, $K$, $k+1$,
                        $\mathcal{V}_{k+1,c}$, $F\|F_{k+1,c}$, $C$, $L$, $E^*$, $F^*$, $\alpha$)
        else
            Calculate expectation $\widehat{E}$ for cascade $F\|F_{k+1,c}$ using (7).
            if $\widehat{E} < \widehat{E}^*$ then
                $\widehat{E}^* := \widehat{E}$,   $F^* := F\|F_{k+1,c}$.
                return $(F^*, \widehat{E}^*)$.
    return $(F^*, E^*)$.
```

4 Experiments

In all experiments we apply *RealBoost+bins* [13] as the main learning algorithm, producing ensembles of weak classifiers as successive cascade stages. Each weak classifier is based on a single selected feature.

Experiments on two collections of images are carried out. Firstly, we test the proposed approach in face detection task, using Haar-like features (HFs) as input information. Secondly, we experiment with synthetic images representing letters (computer fonts originally prepared by T.E. de Campos et al. [5]) and we treat the 'A' letter as our target object. In that experiment we expect to detect our targets regardless of their rotation. To do so, we apply rotationally invariant features based on Zernike moments (ZMs) [2]. In both cases, feature extraction is backed with integral images (complex-valued for ZMs).

In experiments we used a machine with Intel Core i7-4790K 4/8 c/t, 8MB cache. For clear interpretation of time measurements, we report detection times using only a single thread [ST]. The software has been programmed in C#, with key computational procedures implemented in C++ as a dll library.

Experiment: "Faces" (Haar-like features) Training faces were cropped from 3 000 images, looked up using *Google Images*, yielding 7 258 face examples described by 14 406 HFs. The test set contained 3 014 examples from *Essex Face Data* [16]. Validation sets contained 1 000 examples. The number of negatives in the test set was constant and equal to 1 000 000. To reduce training time, the number of negatives in training and validation sets was gradually reduced for successive stages, as described in Table 1. Detection times, reported later, were determined as averages from 200 executions of the detection procedure.

Table 1. "Faces": experimental setup.

train data		**validation data**		**test data**		**detection procedure**	
qty. / parameter	value	qty. / parameter	value	qty. / parameter	value	qty. / parameter	value
no. of positives	7 258	no. of positives	1000	no. of positives	3014	no. of repetitions	200
no. of negatives	139 373	no. of negatives	40 000	no. of negatives	1 000 000	image resolution	600 × 480
" 2nd stage	42 742	" other stages	24 000	total set size	1 003 014	no. of detection scales	5
" other stages	27 742	total set size	41 000			window growing coef.	1.2
total set size	146 631	" other stages	25 000			smallest window	48 × 48
" 2nd stage	50 000					largest window size	100 × 100
" other stages	35 000					window jumping coef.	0.05

We start reporting results by showing some visual examples of detection outcomes obtained by two best detectors (in terms of expected number of features) trained to satisfy $A = 10^{-4}$ and $A = 10^{-5}$ requirements, respectively, see Fig. 2.

Table 2 provides detailed information about cascades trained with $A = 10^{-3}$ requirement. Every row contains a cascade, represented by two sequences: a sequence of features counts n_k on successive stages (top), and a sequence of false alarm rates a_k (bottom). The third column reports the expected value $\widehat{E}(n)$ calculated according to (4). The right-most columns provide information about the effectiveness of tree pruning, showing how many nodes were in fact trained with respect to the potential total. We allow ourselves to report approximate pruning (for both $\alpha = 0.8$ and $\alpha = 1.2$) in the same row as exact pruning, because in all experiments the approximate pruning has never led to a suboptimal solution.

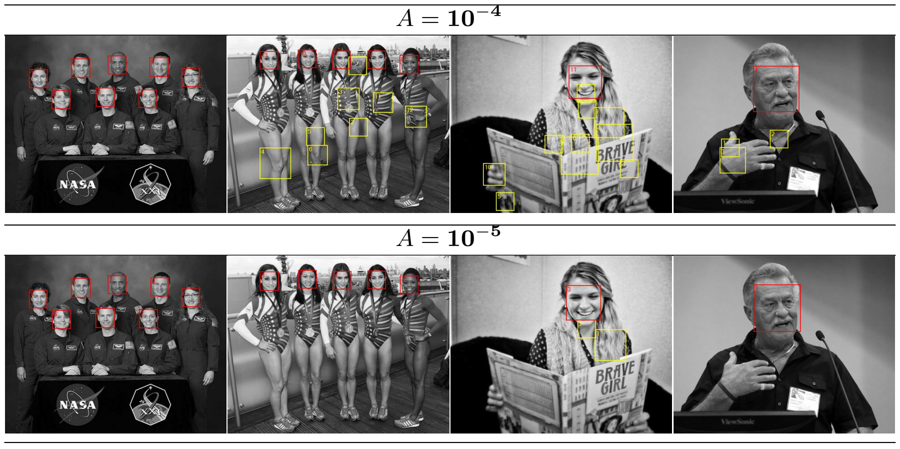

Fig. 2. “Faces”: detection examples (false alarms marked in yellow).

Table 2. “Faces”: cascades trained for $A = 10^{-3}$ (pruning information in last columns).

Training alghorithm	Cascade	$E(n)$	Validation FAR	Validation sensitivity	Trained nodes pruning exact	Trained nodes pruning approximate	
			Requirements: $\mathbf{10^{-3}} =$	**0.9500**		$\alpha = 0.8$	$\alpha = 1.2$
TREE-C3-L1 VJ	8 16 20 23 50 0.2703, 0.2329, 0.2357, 0.2284, 0.2439	14.0932	0.000827	0.9520	11/15	10/15	10/15
TREE-C3-L1 UGM	8 16 20 22 32 0.2703, 0.2329, 0.2357, 0.2324, 0.2781	14.0193	0.000959	0.9510	11/15	10/15	10/15
TREE-C3-L1 UGM-G	8 16 20 22 32 0.2703, 0.2329, 0.2357, 0.2324, 0.2781	14.0193	0.000959	0.9510	11/15	11/15	10/15
TREE-C3-L2 VJ	8 17 16 20 49 0.2703, 0.2152, 0.2442, 0.2445, 0.2483	13.9789	0.000862	0.9520	27/39	24/39	23/39
TREE-C3-L2 UGM	8 17 16 20 46 0.2703, 0.2152, 0.2442, 0.2434, 0.2834	13.9678	0.000980	0.9510	27/39	25/39	23/39
TREE-C3-L2 UGM-G	8 17 16 19 41 0.2703, 0.2152, 0.2442, 0.2776, 0.2406	13.9562	0.000949	0.9510	27/39	26/39	24/39
TREE-C5-L1 VJ	7 18 20 35 40 0.2770, 0.2134, 0.2358, 0.2402, 0.2328	13.7886	0.000779	0.9520	17/25	15/25	14/25
TREE-C5-L1 UGM	7 18 20 31 34 0.2770, 0.2134, 0.2358, 0.2564, 0.2685	13.7204	0.000959	0.9510	17/25	15/25	14/25
TREE-C5-L1 UGM-G	7 18 18 27 53 0.2770, 0.2134, 0.2612, 0.2492, 0.2483	13.6694	0.000955	0.9510	17/25	15/25	13/25
TREE-C3-L1 VJ	3 5 12 4 12 11 16 25 39 38 0.7595, 0.4615, 0.4471, 0.4997, 0.4532, 0.4714, 0.4763, 0.4879, 0.4639, 0.4877	13.6463	0.000880	0.9550	22/30	21/30	19/30
TREE-C3-L1 UGM	4 13 7 9 12 13 12 25 16 17 0.4521, 0.4820, 0.3889, 0.5291, 0.4676, 0.4938, 0.5189, 0.5688, 0.5490, 0.5881	13.3128	0.000987	0.9500	24/30	23/30	22/20
TREE-C3-L1 UGM-G	4 9 10 8 11 12 23 19 33 24 0.4521, 0.5445, 0.4980, 0.4787, 0.5053, 0.5089, 0.4976, 0.4954, 0.4451, 0.5970	13.1655	0.000989	0.9510	23/30	23/30	23/30
TREE-C3-L2 VJ	3 5 12 4 12 11 16 25 39 38 0.7595, 0.4615, 0.4471, 0.4997, 0.4532, 0.4714, 0.4763, 0.4879, 0.4639, 0.4877	13.6463	0.000880	0.9550	51/84	45/84	40/84
TREE-C3-L2 UGM	4 13 7 9 12 13 12 25 16 17 0.4521, 0.4820, 0.3889, 0.5291, 0.4676, 0.4938, 0.5189, 0.5688, 0.5490, 0.5881	13.3128	0.000987	0.9500	52/84	44/84	39/84
TREE-C3-L2 UGM-G	4 10 9 6 13 22 15 14 22 21 0.4521, 0.5038, 0.4470, 0.6043, 0.4975, 0.5153, 0.4436, 0.5666, 0.4644, 0.5352	13.1160	0.000985	0.9510	60/84	57/84	55/84

The table shows clearly that in general the greater the “bushiness” of the tree the better the expected value we try to minimize — an increase in either C or L parameter lead to an improvement. Additionally, owing to pruning, the time needed to train cascades involving wider trees did not increase proportionally to the overall number of nodes. One should realize that nodes (stages) lying deeper in the tree, with low effective FAR resulting from chain multiplcation of a_k rates, require much time for resampling, since only a small fraction of negative examples reaches those stages. That is why it is so important to prune redundant nodes. In particular, for TREE-C3-L2-UGM-G an exhaustive search would re-

Table 3. “Faces”: VJ vs tree-based cascades with K=5 (left) and K=10 (right) stages.

Training alghorithm	$E(n)$	Validation FAR	Validation sensitivity	Test FAR	Test sensitivity	$\bar{n}$	Detection time image [ST][ms]	Detection time window [ST][μs]
		Requirements: **10^{-3}**	**0.9500**				(windows per image: 130 971)	
VJ	14.97	0.000792	0.9520	0.000815	0.9549	15.88	83	0.64
TREE C5-L1 UGM-G	13.67	0.000955	0.9510	0.001078	0.9668	14.59	73	0.56
		Requirements: **10^{-4}**	**0.9500**				(windows per image: 130 971)	
VJ	23.88	0.000091	0.9520	0.000107	0.9482	25.80	125	0.95
TREE C3-L1 UGM-G	22.21	0.000097	0.9510	0.000099	0.9542	23.88	115	0.87

Training alghorithm	$E(n)$	Validation FAR	Validation sensitivity	Test FAR	Test sensitivity	$\bar{n}$	Detection time image [ST][ms]	Detection time window [ST][μs]
		Requirements: **10^{-3}**	**0.9500**				(windows per image: 130 971)	
VJ	13.78	0.000950	0.9600	0.001048	0.9569	15.78	75	0.57
TREE C3-L2 UGM-G	13.12	0.000985	0.9510	0.001155	0.9545	14.71	71	0.55
		Requirements: **10^{-4}**	**0.9500**				(windows per image: 130 971)	
VJ	15.14	0.000077	0.9550	0.000073	0.9562	16.45	89	0.68
TREE C3-L1 UGM-G	14.38	0.000099	0.9510	0.000108	0.9552	15.29	83	0.63
		Requirements: **10^{-5}**	**0.9500**				(windows per image: 130 971)	
VJ	17.52	0.000006	0.9550	0.000010	0.9482	19.45	100	0.77
TREE C3-L1 UGM-G	17.28	0.000010	0.9510	0.000010	0.9317	19.01	92	0.70

quire 84 nodes, exact pruning reduces this number to 60, whereas approximate pruning cuts it further down to 57 (for $\alpha = 0.8$) and 55 (for $\alpha = 1.2$).

Table 3 compares cascades trained traditionally (VJ) against selected best cascades trained via tree search. The comparison pertains to accuracy and detection times. This time we show three variants of A requirement: 10^{-3}, 10^{-4} and 10^{-5} (that last setting only for cascades with 10 stages). In addition, the theoretical expected value for cascades can be compared against an avarage observed on the test set (column $\bar{n}$). We remark that the tree-based approach combined with greedy per-stage requirements — TREE-C3-L1-UGM-G — produced the best cascades (marked with dark gray) having the smallest expected values. Savings in detection times per image with respect to VJ approach are at the level of $\approx$ 7.5 ms (about 8% per thread). This may seems not large but we remind that the measurements are for single-threaded executions [ST]. For example, if 8 threads are used this implies a reduction of $\approx$ 4 FPS.

Experiment: “Synthetic A letters” (Zernike Moments) Tab. 4 lists details of the experimental setup for this experiment. In train images, only objects with limited rotations were allowed ($\pm 45°$ with respect to their upright positions). In contrast, in test images, rotations within the full range of 360° were allowed. During the training 540 features were at disposal [2].

Table 4. “Synthetic A letters”: experimental setup.

train data		**validation data**		**test data**		**detection procedure**	
qty. / parameter	value	qty. / parameter	value	qty. / parameter	value	qty. / parameter	value
no. of positives	20 384	no. of positives	1 000	no. of positives	20 000	no. of repetitions	200
no. of negatives	50 546	no. of negatives	10 000	no. of negatives	1 000 000	image resolution	600 × 480
total set size	70 930	total set size	11 000	total set size	1 020 000	no. of detection scales	5
						window growing coef.	1.2
						smallest window	100 × 100
						largest window size	208 × 208
						window jumping coef.	0.05

Fig. 3 presents examples of detection outcomes obtained by best detectors trained to satisfy 10^{-3} and 10^{-4} FAR requirements. As it turned out for this data, the cascades did not need many stages nor features. Table 5 compares VJ against tree-based cascades. One can note that despite small feature counts (comparing to the previous experiment), the proposed method still allows to reduce the expectations. The smallest were achieved by the TREE-C3-L1-UGM-G variant, yielding 2.5682 and 2.9910, respectively for $A = 10^{-3}$ and $A = 10^{-4}$.

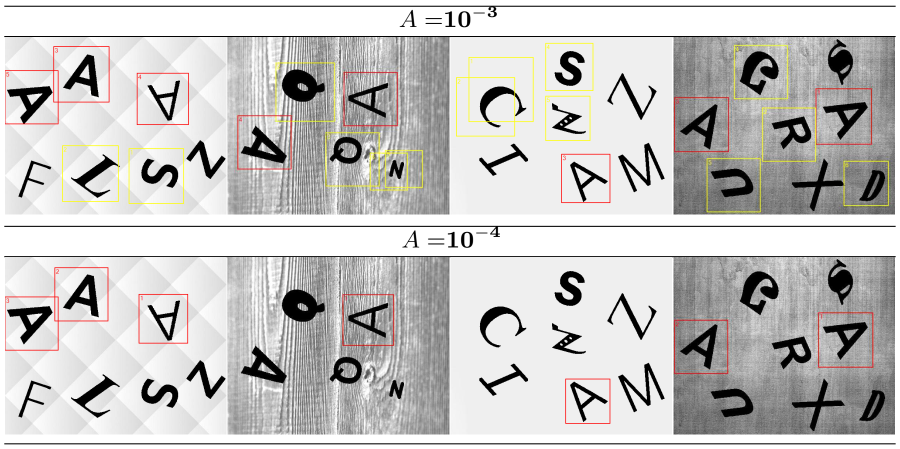

Fig. 3. "Synthetic A letters": detection examples.

Table 5. "Synthetic A letters": VJ vs tree-based cascades with $K = 5$ stages.

Training alghorithm	Cascade	Expected value	Validation		Test			Detection time	
			FAR	sensiti-vity	FAR	sensiti-vity	$\bar{n}$	image [ST][ms]	window [ST][μs]
			Requirements: **10^{-3}**	**0.9500**				(windows per image: 18 752)	
VJ	$\overset{2}{0.2115}, \overset{2}{0.2128}, \overset{3}{0.1893}, \overset{5}{0.2180}, \overset{7}{0.2098}$	2.6136	0.000389	0.9540	0.000869	0.9313	2.56	127	6.77
TREE-C3-L1-UGM-G	$\overset{2}{0.2115}, \overset{2}{0.2128}, \overset{2}{0.2954}, \overset{3}{0.2313}, \overset{5}{0.2812}$	2.5682	0.000865	0.9500	0.002016	0.9351	2.50	124	6.61
			Requirements: **10^{-4}**	**0.9500**				(windows per image: 18 752)	
VJ	$\overset{3}{0.1497}, \overset{4}{0.1460}, \overset{5}{0.1444}, \overset{8}{0.1368}, \overset{10}{0.1206}$	3.7393	0.000052	0.9530	0.000163	0.9411	3.84	157	8.37
TREE-C3-L1-UGM-G	$\overset{2}{0.2115}, \overset{4}{0.1137}, \overset{5}{0.1284}, \overset{6}{0.1824}, \overset{11}{0.1535}$	2.9910	0.000086	0.9510	0.000203	0.9418	2.99	133	7.10

5 Conclusion

Training a cascade of classifiers is a difficult optimization problem that, in our opinion, should be always carried out with a primary focus on the *expected number of extracted features*. This quantity reflects directly how fast an operating cascade is. Our proposition of the tree search-based training allows to 'track' more than one variant of a cascade. Potentially, this approach can be computationally expensive, but we have managed to reduce it with suitable branch-and-bound techniques. Being able to prune some of the subtrees, we save both the training and resampling time needed by later cascade stages. To our knowledge, no such proposition regarding the cascade structure has been tried out before. In our future research we plan to investigate more the approximate variant, trying to predict partial expectations for more than one stage ahead.

References

1. Abbas, S., et al.: Crowd detection and management using cascade classifier on armv8 and opencv-python. In: 2017 Intern. Conf. on Innovations in Information, Embedded and Communication Systems (ICIIECS). pp. 1–6 (March 2017)
2. Bera, A., Klęsk, P., Sychel, D.: Constant-Time Calculation of Zernike Moments for Detection with Rotational Invariance. IEEE Transactions on Pattern Analysis and Machine Intelligence **41**(3), 537–551 (2019)
3. Bourdev, L., Brandt, J.: Robust Object Detection via Soft Cascade. In: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2. pp. 236–243. IEEE Computer Society (2005)
4. Budiman, R.A.M., Achmad, B., Faridah, Arif, A., Nopriadi, Zharif, L.: Localization of white blood cell images using haar cascade classifiers. In: 2016 1st International Conference on Biomedical Engineering (IBIOMED). pp. 1–5 (Oct 2016)
5. de Campos, T.E., et al.: Character recognition in natural images. In: Intern. Conf. on Computer Vision Theory and Applications, Portugal. pp. 273–280 (2009)
6. C.H., S., et al.: Thermal image human detection using haar-cascade classifier. In: 2017 7th Intern. Annual Engineering Seminar (InAES). pp. 1–6 (2017)
7. J., L., et al.: Detection of bird's nest in high power lines in the vicinity of remote campus based on combination features and cascade classifier. IEEE Access **6**, 39063–39071 (2018)
8. Jones, N., Pevzner, P.: An Introduction to Bioinformatics Algorithms. MIT Press (2002)
9. L., C., et al.: Human face detection algorithm via haar cascade classifier combined with three additional classifiers. In: 2017 13th IEEE Intern. Conf. on Electronic Measurement Instruments (ICEMI). pp. 483–487 (2017)
10. Li, J., Zhang, Y.: Learning SURF Cascade for Fast and Accurate Object Detection. In: Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition. pp. 3468–3475. CVPR '13, IEEE Computer Society (2013)
11. Li, Y., Xu, X., Mu, N., Chen, L.: Eye-gaze tracking system by haar cascade classifier. In: 2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA). pp. 564–567 (June 2016)
12. Pham, M., Cham, T.: Fast training and selection of Haar features using statistics in boosting-based face detection. In: Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on. pp. 1–7 (2007)
13. Rasolzadeh, B., et al.: Response Binning: Improved Weak Classifiers for Boosting. In: IEEE Intelligent Vehicles Symposium. pp. 344–349 (2006)
14. Saberian, M., Vasconcelos, N.: Boosting Algorithms for Detector Cascade Learning. Journal of Machine Learning Research **15**, 2569–2605 (2014)
15. Shen, C., Wang, P., Paisitkriangkrai, S., van den Hengel, A.: Training Effective Node Classifiers for Cascade Classification. International Journal of Computer Vision **103**(3), 326–347 (2013)
16. University of Essex: Face Recognition Data. https://cswww.essex.ac.uk/mv /allfaces/faces96.html (1997), [Online; accessed 11-May-2019]
17. Vallez, N., Deniz, O., Bueno, G.: Sample selection for training cascade detectors. PLos ONE **10** (2015)
18. Viola, P., Jones, M.: Robust Real-time Face Detection. International Journal of Computer Vision **57**(2), 137–154 (2004)

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50423-6_2