Relaxed Per-Stage Requirements for Training Cascades of Classifiers

Dariusz Sychel and Przemysław Klęsk and Aneta Bera

Abstract. Historically first training algorithms for cascades of classifiers were guided by constant per-stage requirements (constraints) imposed on false alarm and detection rates. Those constant values were calculated according to the geometric mean rule, implied by a pair of final requirements predefined for the whole cascade.

We provide and prove a theoretical result demonstrating that the presence of slack between the constant requirements and actual rates observed while learning, allows to introduce new relaxed requirements for each successive stage and still complete the training procedure successfully (with final requirements satisfied). The relaxed requirements can be met more easily, using fewer features. This creates a potential possibility to reduce the expected number of features used by an operating cascade — the crucial quantity we focus on in the paper. Taking advantage of the relaxation, we propose new stage-wise training algorithms that apply two approaches: uniform or greedy. They differ in the way the slack cumulated so far becomes “consumed” later on.

Reported experimental results pertain to cascades trained to work as face or letter detectors, with Haar-like features or Zernike moments being the input information, respectively. The results confirm shorter operating times of cascades obtained by the proposed technique, owing to the reduction in the number of extracted features.

1 Introduction

Cascades of classifiers were in principle designed to work as classifying systems that operate under the following specific conditions: (1) very large number of incoming requests, (2) significant imbalance of classes. The natural scenario where these two conditions appear are detection procedures. Images, video sequences or sound data can be analyzed in order to detect some target objects or events of interest. Typically, detection procedures perform dense data scans using a sliding window, and thereby generate thousands of requests for classification in short periods of time. Apart from computer vision, cascades can be also applied in various batch classification jobs. In such scenarios, we expect the classification system to process in the background large portions of data incoming on regular basis, e.g.: medical samples, shopping transactions, satellite photos, system logs, etc.

The second condition pointed out — imbalance of classes — should not be seen as a difficulty but rather a favorable setting that makes the whole concept viable. Namely, a cascade should vary its computational efforts depending on the contents of an object to be classified. Obvious negatives (non-targets) should be recognized fast, using only a few extracted features. This is because negative objects constitute a vast majority of all objects (e.g. background regions in computer vision applications). On the other hand, positive objects (targets) or the ones resembling them, should be allowed to employ more time and computations based on hundreds or even thousands of features.

There exist a certain average value of the computational cost incurred by an operating cascade (in between the two mentioned extremes). This quantity can be defined mathematically as an expected value (we do this in Section 2.3) and, in fact, calculated explicitly for a given cascade in terms of:
- number of features applied on successive stages,
- false alarm rates on successive stages,
- detection rates (sensitivities) on successive stages,
- probability distribution from which the data is drawn.

Since the true probability distribution underlying the data is typically unknown in practice, the exact expected value cannot be determined. Interestingly though, it can be very accurately approximated using just the first two pieces of information in the list.

Training procedures for cascades are time-consuming. For a suitably complex problem the training may take days or even weeks to complete. As Viola and Jones noted in their pioneering works [22, 23], cascade training is a difficult combinatorial optimization problem which involves the following parameters: number of cascade stages, number of features on successive stages, selection of those features, and finally decision thresholds. It is worth to remark that this problem has not been ultimately solved yet, Viola and Jones tackled it by imposing: the number of stages (say $K$ stages) and the final requirements the whole cascade should meet in order to be accepted by the user. These requirements were 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 apply the geometric mean to translate the final requirements onto another pair of numbers — per-stage requirements — $(a_{\text{max}}, d_{\text{min}})$, where $a_{\text{max}} = A^{1/K}$ and $d_{\text{min}} = D^{1/K}$. If each stage satisfies such requirements then the whole cascade also satisfies the final requirements.

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 [9, 4, 13, 10, 21]. It is fair to remark that Haar-like, HOG and LBP features are by far the most common ones in literature. Some authors obtain modified cascades by designing new boosting algorithms that underlie the training process [18, 17, 15].

In our view, a particularly elegant algorithm was proposed by Saberian and Vasconcelos [15]. The authors try to optimize explicitly a Lagrangian representing the trade-off between cascade’s error rate and the operating computational cost. The approach is based on
recursive formulas (both for the cascade response and the number of applied features) that are translated from discrete non-differentiable versions to smooth ones using a certain mathematical trick. The trick replaces indicator functions (i.e. zero-one functions that signal if a cascade stage is executed or not) by hyperbolic tangent functions that are smooth. In consequence, the gradient descent optimization can be performed. The resulting training procedure is not stage-wise. All stages are potentially kept open. Each gradient step can either: add a new weak classifier to any of the stages, or create a new stage by cloning the last existing one (operation proven to be neutral). The whole approach is analytically tractable, which is a great property, but there are some drawbacks too. First of all, the gradient descent can get stuck in local optima [19]. Secondly, it is difficult to guess a good value for the Lagrange multiplier. Last, but not least, the performed optimization is in a sense exhaustive. One has to enumerate a few examples: crowd analysis and people counting [1], electric lines [12].

Let us enumerate a few examples: crowd analysis and people counting [1], detection of birds near high power electric lines [12].

Consistent training algorithms based on the so-called relaxed per-stage requirements. Such requirements can be introduced by observing the actual rates (false alarm rate and detection rate) obtained in the progress of the algorithm, and the slack created by them with respect to final requirements. In Section 3 we prove a theorem that motivates the usage of relaxed requirements. We analyze some properties implied by this theorem and indicate how they can reduce the expected number of stages.

The main contribution of this paper are two new cascade training algorithms based on the so-called relaxed per-stage requirements. Such requirements can be introduced by observing the actual rates (false alarm rate and detection rate) obtained in the progress of the algorithm, and the slack created by them with respect to final requirements. In Section 3 we prove a theorem that motivates the usage of relaxed requirements. We analyze some properties implied by this theorem and indicate how they can reduce the expected number of stages.

The probabilistic meaning of relevant quantities above is as follows. The final requirement on false alarm rate means that:

$$P(F(x) = + | y = -) \leq A.$$  (1)

The final requirement on detection rate (sensitivity) means that:

$$P(F(x) = + | y = +) \geq D.$$  (2)

Actual false alarm and detection rates observed during the training procedure are, respectively, equal to

$$a_k = 1 - P(F_k(x) = - | y = -), F_k(x) = \cdots = F_{k-1}(x) = +) = P(F_k(x) = + | y = -), F_1(x) = \cdots = F_{k-1}(x) = +),$$

$$d_k = P(F_k(x) = + | y = +), F_1(x) = \cdots = F_{k-1}(x) = +).$$  (3)

### 2.2 Classical cascade training (Viola-Jones style)

In algorithmic pseudocodes to follow in this paper we shall use the $\parallel$ symbol to denote concatenation of cascade stages. For example, when the current cascade is $F = (F_1, \ldots, F_K)$ then the notation $F \parallel F_{k+1}$ should be understood as an extended cascade $(F_1, \ldots, F_k, F_{k+1})$, but having no effect on $F$ so far. Whereas, $F := F \parallel F_{k+1}$ means that the next stage has been in fact appended to the cascade, so that in effect it becomes $F = (F_1, \ldots, F_k, F_{k+1})$.

The classical cascade training algorithm presented below (Algorithm 1) can be treated as reference for new propositions given in this paper. 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 early, when $k < K$, provided that the final requirements $(A, D)$ for the entire cascade are already satisfied i.e. $A_k \leq A$ and $D_k \geq D$.

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\(^2\)

$$F_k(x) - \theta_k,$$

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

$$\theta_{k+1} = v_{(1-d_{\min}) \#P}.$$  (4)

\(^2\) Zero value can be arbitrarily mapped to $-1$ or $+1$
Algorithm 1 Cascade of classifiers training algorithm with constant per-stage requirements

```
procedure TRAIN(VJ/CASCADE(D, A, D, K, V))
From D take subset P with all positive examples, and subset N with all negative examples.
F := () \(\triangleright\) initial cascade — empty sequence
a_{max} := A^{1/K}, a_{min} := D^{1/K}. \(\triangleright\) constant requirements
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.
    \(a_{k+1} := \frac{A_{k+1}}{A_k}.\)
    while \(a_{k+1} > a_{max}\) do
        \(n_{k+1} := n_{k+1} + 1.\)
        Train new weak classifier \(f\) using \(P\) and \(N\)
        \(F_{k+1} := F_{k+1} + f.\)
        Adjust decision threshold \(\theta_{k+1}\) for \(F_{k+1}\)
        to satisfy \(a_{min}\) requirement.
        Use cascade \(F[F_{k+1}]\) on validation
        set \(V\) to measure \(A_{k+1}\) and \(D_{k+1}.
        a_{k+1} := \frac{A_{k+1}}{A_k}.
        F := F[F_{k+1}].\)
if \(A_{k+1} > A\) then
    \(N := \emptyset.\)
    Use current cascade \(F\) to populate \(N\) with
    false detections sampled from non-face images.
    \(k := k + 1.\)
return \(F = (F_1, F_2, \ldots, F_k).\)
```

2.3 Expected number of extracted features

2.3.1 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: \(n_1, n_1 + n_2, \ldots, n_1 + n_2 + \cdots + n_K.\) By the definition of expected value, the expected number of features can be calculated as follows:

\[
E(n) = \sum_{1 \leq k \leq K} \left( \sum_{1 \leq i \leq k} n_i \left( p \left( \prod_{1 \leq i \leq k} d_i \right) \left(1 - d_k \right)^{k<K} \right)
+ (1 - p) \left( \prod_{1 \leq i < k} a_i \right) \left(1 - a_k \right)^{k<K} \right).
\]

where \([\cdot]\) is an indicator function.

2.3.2 Incremental formula and its approximation

By grouping the terms in (5) with respect to \(n_k\) the following alternative formula can be derived:

\[
E(n) = \sum_{1 \leq k \leq K} n_k \left( p \left( \prod_{1 \leq i < k} d_i \right) + (1 - p) \left( \prod_{1 \leq i < k} a_i \right) \right).
\]

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

\[
\tilde{E}(n) = \sum_{1 \leq k \leq K} n_k \prod_{1 \leq i < k} a_i \approx E(n).
\]

It is also interesting to remark that in the original Viola and Jones’ paper [23] the authors proposed an incorrect formula to estimate the expected number of features, namely:

\[
E_{VJ}(n) = \sum_{k=1}^{K} n_k \prod_{i=1}^{k-1} r_i, \tag{8}
\]

where \(r_i\) represents the “positive rate” of \(i\)-th stage. This is equivalent to

\[
E_{VJ}(n) = \sum_{k=1}^{K} n_k \prod_{i=1}^{k-1} (pd_i + (1-p)a_i), \tag{9}
\]

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 Motivation theorem

The following theorem points out the possibility of relaxation of per-stage requirements and thereby constitutes our base motivation to propose new variations of cascade training algorithms.

Theorem 1 The presence of slack between constant per-stage requirements \((a_{max}, a_{min})\) and actual rates \((a_k, d_k)\), for \(k = 1, \ldots, K\), observed during cascade training —

\[
a_k = (1 - \epsilon_k)a_{max}, \quad d_k = (1 + \delta_k)a_{min}, \tag{10}
\]

where \(\epsilon_k, \delta_k\) represent slack variables denoting small numbers — allows to introduce new relaxed requirements for each successive stage and carry out a training procedure that still satisfies the final requirements \((A, D)\) for the whole cascade. In particular, when the \(k\)-th stage is done, the following two pairs of relaxed bounds (uniform and greedy) can be applied for the \((k + 1)\)-th stage:

\[
a_{k+1} \leq \frac{a_{max}}{(1 - \epsilon_k)^{1/(K+1)}}, \quad d_{k+1} \geq \frac{d_{min}}{(1 + \delta_k)^{1/(K+1)}}, \tag{11}
\]

or

\[
a_{k+1} \leq \frac{a_{max}}{1 - \epsilon_k}, \quad d_{k+1} \geq \frac{d_{min}}{1 + \delta_k}, \tag{12}
\]

where \(1 - \epsilon_k = \prod_{1 \leq i \leq k} (1 - \epsilon_i)\) and \(1 + \delta_k = \prod_{1 \leq i \leq k} (1 + \delta_i).\)

Before going into the proof, the following remarks should be made. The theorem states that slack variables \(\epsilon_k, \delta_k\) are small numbers, but it purposely does not specify their sign. Intuitively it seems that \(\epsilon_k, \delta_k \geq 0.\) But as a matter of fact, when relaxed bounds are applied throughout the training procedure, some of \(\epsilon_k, \delta_k\) can become negative too. To see this, consider for example the first two variables \(\epsilon_1, \epsilon_2\) and the bound (12). It implies that \(a_2 = (1 - \epsilon_2)a_{max} \leq a_{max}/(1 - \epsilon_1)\) and therefore:

\[
\epsilon_2 \geq \frac{-\epsilon_1}{1 - \epsilon_1}. \tag{13}
\]

Simultaneously note that \(\epsilon_1 \geq 0\) since \(a_1 = (1 - \epsilon_1)a_{max}\) must not exceed the original bound \(a_{max}\). Therefore, (13) informs that \(\epsilon_2\)
may, in particular, be negative. What is important though, is that the ‘effective’ slack variables $\epsilon_{\leq k}$, $\delta_{\leq k}$ (resulting from the product rules) should be guaranteed to be non-negative, so that the relaxation in bounds (11), (12) indeed takes place. The following lemma states this fact and is further used to prove the main theorem.

**Lemma 1** $\epsilon_{\leq k} \geq 0$ and $\delta_{\leq k} \geq 0$ for all $k = 1, \ldots, K$ and regardless of the bound applied: uniform (11) or greedy (12).

**Proof.** It suffices to prove the lemma only for $\epsilon_{\leq k}$ variables, the arguments are analogous for $\delta_{\leq k}$ with direction of all inequalities reversed. We first look at the uniform bound (11) and prove the lemma by induction. Base: $\epsilon_{1} = \epsilon_{1} \geq 0$ follows from the bound on $a_{1} = (1 - \epsilon_{1})a_{\text{max}} \leq a_{\text{max}}/\prod_{1 \leq i \leq 0}(1 - \epsilon_{i})^{1/(K-0)}$ since the empty product in the denominator yields 1. Inductive step: suppose the lemma is true for all indexes up to $k$, i.e. $\epsilon_{\leq k+1} \geq 0$. Then, to see that $\epsilon_{\leq k+1} \geq 0$, it suffices to show that $1 - \epsilon_{k+1} = (1 - \epsilon_{k})(1 - \epsilon_{k+1}) \leq 1$. From bound (11) we have that:

$$
\begin{align*}
\epsilon_{k+1} &\leq \frac{a_{\text{max}}}{(1 - \epsilon_{k})^{1/(K-k)}} \\
(1 - \epsilon_{k+1})a_{\text{max}} &\leq \frac{a_{\text{max}}}{(1 - \epsilon_{k})^{1/(K-k)}} \\
(1 - \epsilon_{k+1})(1 - \epsilon_{k})^{1/(K-k)} &\leq 1 \\
(1 - \epsilon_{k+1})(1 - \epsilon_{k}) &\leq 1 \\
1 - \epsilon_{k+1} &\leq 1.
\end{align*}
$$

(14)

The third pass is true because the inductive assumption $\epsilon_{i} \geq 0$ implies that expression $(1 - \epsilon_{i})^{1/(K-k)}$ is a fraction raised to a fractional power. Hence, omitting the power lowers the left-hand-side. This proves the lemma correctness for the uniform bound. It is easy to check that the inductive step for the greedy bound (12) leads directly to inequality: $(1 - \epsilon_{k+1})(1 - \epsilon_{k}) \leq 1$ and the base is satisfied as well.

**Proof of Theorem 1.** By virtue of lemma 1, note that when $\epsilon_{\leq k}, \delta_{\leq k} > 0$ the denominators in both (11) and (12) cause that new per-stage requirements can be met more easily than original ones. We now limit the considerations only to the sequence of false alarm rates $a_{1}, \ldots, a_{K}$ (the arguments are analogical for detection rates). Suppose $k$ training stages are already done. To satisfy the final requirement the following inequality must hold

$$
(1 - \epsilon_{1})a_{\text{max}} \cdots (1 - \epsilon_{k})a_{\text{max}}^{K-k} a_{k+1} \cdots a_{K} \leq a_{\text{max}}^{K}
$$

for $k$ initial stages

$$
(1 - \epsilon_{k+1})a_{\text{max}}^{K-k} a_{k+1} \cdots a_{K} \leq a_{\text{max}}^{K-k}(1 - \epsilon_{k}).
$$

(15)

Now, it is possible to see the two approaches to bound the remaining rates $a_{k+1}, \ldots, a_{K}$. The first is to let all of them consume uniformly the slack ‘cumulated’ so far $\prod_{1 \leq i \leq k}(1 - \epsilon_{i}) = 1 - \epsilon_{\leq k}$. To do so, think of a mean factor $a_{\alpha}$ such that $a_{\alpha}^{K-k} = a_{k+1} \cdots a_{K}$ in (15). Now, the root of order $K - k$ taken sideways yields formula (11) — the uniform bound. This approach can be interpreted as an updated geometric mean on remaining false alarm rates. The second approach is to let the very next rate $a_{k+1}$ consume all the slack and assume pessimistically the subsequent rates $a_{k+2}, \ldots, a_{K}$ to be equal to the original bound $a_{\text{max}}$. This leads to inequality $a_{k+1}a_{\text{max}}^{K-k-1} \leq \frac{a_{\text{max}}^{K-k}}{(1 - \epsilon_{k})}$ and yields formula (12) — the greedy bound.

As regards the second approach, its greediness can be also well understood through the following consequence of relaxed bounds (12).

**Corollary 1** If $(k+1)$-th stage becomes trained according to the greedy approach and the observed resulting $a_{k+1}$ is not less but exactly equal to the bound $a_{\text{max}}/(1 - \epsilon_{k})$, then the next stage requirement for $a_{k+2}$ tightens back to the original bound $a_{\text{max}}$.

**Proof.** The result follows directly from (15) by: isolating out $a_{k+2}$, inserting the right hand side of (12) into $a_{k+1}$ and setting $a_{1} = a_{\text{max}}$ for all $i \geq k + 3$.

The same argument is true for any two consecutive sensitivities $d_{k+1}, d_{k+2}$.

Can relaxed per-stage requirements reduce the expected number of extracted features?

Since relaxed per-stage requirements can be satisfied more easily — with fewer features — one might be tempted to think that this leads directly to a reduction of $\hat{E}(n)$. Unfortunately, this is not true in general. Even though $n_{k}$ numbers in (7) can in fact be decreased in many cases, note that the cost paid for that is an increase of false alarm rates $a_{\alpha}$ that are also present in formula (7). Moreover, note that each increased $a_{\alpha}$ contributes in a multiplicative manner to all subsequent summations in the expectation. Therefore, we remark that the relaxation provides the necessary (but not sufficient) condition to reduce the expected value.

## 4 Relaxed cascade training

### 4.1 Relaxed per-stage requirements expressed without slack variables

At implementation level there is no need to explicitly calculate the slack variables. The new per-stage requirements can be expressed in terms of $A, D$ constants and $a_{i}, d_{i}$ rates observed so far, that is for $i \leq k$. It is easy to check that the following pairs of formulas are equivalent counterparts of the right-hand-sides of (11) and (12), respectively.

$$
\begin{align*}
A_{\text{min},k+1} &= \left(\frac{A}{\prod_{1 \leq i \leq k} a_{i}}\right)^{\frac{1}{\epsilon_{k+1}}} , & d_{\text{min},k+1} &= \left(\frac{D}{\prod_{1 \leq i \leq k} d_{i}}\right)^{\frac{1}{\epsilon_{k+1}}} .
\end{align*}
$$

(16)

(17)

### 4.2 Training algorithm

Formulas (16) and (17) can be directly applied to form variations of the classical VJ cascade training procedure (Algorithm 1). It suffices to use them as replacements of constant requirements ($a_{\text{max}}, d_{\text{min}}$) in every iteration of the main loop. Algorithm 2 demonstrates the ‘relaxed’ cascade training procedure.

In the experimental section, we shall refer to Algorithm 2 coupled with formula (16) by the name UGM (standing for: Updated Geometric Mean), whereas for Algorithm 2 coupled with (17) we shall use the name UGM-G (Updated Geometric Mean – Greedy).
Algorithm 2 Cascade of classifiers training algorithm with relaxed per-stage requirements

procedure TRAINRELAXEDCASCADE(\(\mathcal{D}, A, D, K, V\))

From \(\mathcal{D}\) take subset \(\mathcal{P}\) with all positive examples, and subset \(\mathcal{N}\) with all negative examples.

\(\mathcal{F} := (\emptyset, D_0 := 1, k := 0)\) \(\triangleright\) initial cascade — empty sequence

while \(\mathcal{A}_k > \mathcal{A}_0\) do

\(F_{k+1} := \text{TRAINSTAGE}(\mathcal{P}, \mathcal{N}, K, k, \mathcal{V}, \mathcal{F})\).

\(\mathcal{F} := \mathcal{F} | F_{k+1}\).

if \(\mathcal{A}_{k+1} > \mathcal{A}_0\) then

\(\mathcal{N}^\prime := \emptyset\).

Use current cascade \(\mathcal{F}\) to populate set \(\mathcal{N}^\prime\) with false detections sampled from non-face images.

\(k := k + 1\)

return \(\mathcal{F} = (F_1, F_2, \ldots, F_k)\).

procedure TRAINSTAGE(\(\mathcal{P}, \mathcal{N}, K, k, \mathcal{V}, \mathcal{F}\) )

\(n_{k+1} := 0, D_{k+1} := 0, A_{k+1} := A_k\).

Calculate relaxed per-stage requirements \((\delta_{\text{max}, k+1}, d_{\text{min}, k+1})\) using (16) or (17).

\(a_{k+1} := A_{k+1} / A_k\).

while \(a_{k+1} > \alpha_{\text{max}, k+1}\) 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 \(d_{\text{min}, k+1}\) requirement.

Use cascade \(\mathcal{F} | F_{k+1}\) on validation set \(\mathcal{V}\) to measure \(A_{k+1}\) and \(D_{k+1}\).

\(a_{k+1} := A_{k+1} / A_k\).

return \(F_k\).

5 Experiments

5.1 Learning algorithm and general settings

In all experiments we apply RealBoost+bins [14] as the main learning algorithm, producing ensembles of weak classifiers as successive cascade stages. Each weak classifier is based on a single selected feature. The response of such a classifier is real-valued, calculated according to the logit transform with a binning mechanism.

Experiments on two collections of images are carried out. Firstly, we test the proposed approach in face detection task, using Haar-like features as the input information. Secondly, we experiment with synthetic images representing letters (computer fonts originally prepared by T.E. de Campos et al. [7]). The letters are placed randomly on a set of backgrounds and we treat the ‘A’ letter as our target object. In that second group of experiments we expect to detect our targets regardless of their rotation (rather than in upright position). To do so, we apply rotationally invariant features based on Zernike moments (ZMs) as the input information [3]. In both cases, feature extraction is backed with integral images (complex-valued for ZMs).

In all experiments we used a machine with Intel Core i7-4790K 4/8 cores/threads, 8MB cache. For clear interpretation of time measurements, we report detection times using only a single thread [ST]. The software has been programmed in C8, with key computational procedures implemented in C++ as a dll library.

5.2 “Face detection” — Haar-like features

Training faces were cropped from 3000 images, looked up using Google Images search engine. The training set contained 7 258 face examples. We preset our features space to contain 14 406 Haar-like features\(^3\). The test set contained 3 014 faces from Essex facial images collection [20, 21] and validation sets contained 1 000 face examples. The number of negatives in the test set was constant and equal to 1 000 000. In order to reduce training time, the number of negatives in training and validation sets was gradually reduced for successive cascade stages, as described in Table 1. Detection times of different cascades were determined as averages from 1000 executions.

### Table 1. “Face detection”: experimental setup.

<table>
<thead>
<tr>
<th>Training data</th>
<th>Validation data</th>
<th>Test data</th>
</tr>
</thead>
<tbody>
<tr>
<td>faces scanned with 1 000 windows and 50 000 images</td>
<td>faces scanned with 1 000 windows and 50 000 images</td>
<td>faces scanned with 1 000 windows and 50 000 images</td>
</tr>
<tr>
<td>face detection procedure (with a sliding window)</td>
<td>face detection procedure (with a sliding window)</td>
<td>face detection procedure (with a sliding window)</td>
</tr>
</tbody>
</table>

Fig. 1 shows visual examples of detection outcomes obtained by two best detectors (in terms of the expected number of features), trained to satisfy \(A = 10^{-4}\) and \(A = 10^{-5}\) FAR requirements.

Tables 3, 4 constitute a detailed comparison of all cascades obtained in face detection experiments, respectively for \(K = 5\) and \(K = 10\) stages. The final FAR requirements \((A)\) were imposed to be either: \(10^{-3}\), \(10^{-4}\) or \(10^{-5}\) (that last setting only for cascades with 10 stages). Every row in the tables corresponds to some cascade, represented by two sequences: a sequence of feature counts \(n_k\) (top), and a sequence of false alarm rates \(a_k\) (bottom). Apart from accuracy measures, we report for each cascade its theoretical expected value \(\hat{E}(n)\) calculated according to formula (7). This value can be compared against an average observed on the test set — column \(\text{avg}(n)\).

Let us comment now on results for cascades with \(K = 5\) stages. First of all it is worth noting that all trained cascades satisfied the imposed final requirements. As Table 3 shows, the relaxed greedy bound — UGM-G — produced the best cascades (marked with dark gray), having the smallest expectations: 14.7220 and 23.9587 respectively for \(A = 10^{-3}\) and \(A = 10^{-4}\). The second-best approach (light gray) was not consistent — for \(A = 10^{-3}\), it was the classical VJ cascade yielding the expectation 15.3571, whereas for \(A = 10^{-4}\) it was UGM yielding 24.9455.

As regards cascades with \(K = 10\) stages, results reported in Table 4 indicate similar tendencies. Again, the smallest expectations were achieved by the UGM-G cascades trained using the relaxed

---

\(^3\) generated for 6 waveform templates, \(\tau^2\) scales and \(\tau^2\) anchoring points
greedy bound, regardless of the FAR requirements. The UGM cascade placed itself second in all experiments.

5.3 “Synthetic A letters” — Zernike moments

We remind that a synthetic data set containing capital letters from the modern English alphabet [7] was prepared for this experiment. Fig. 2 gives an overview on the source graphical material, whereas Tab. 2 lists details of the experimental setup.

Table 2. “Synthetic A letters”: experimental setup.

<table>
<thead>
<tr>
<th>quantity / parameter</th>
<th>value / additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>no. of positive examples</td>
<td>20,384 windows with letter ‘A’</td>
</tr>
<tr>
<td>no. of negative examples</td>
<td>70,930 windows with letters other than ‘A’ plus random samples of backgrounds on each stage after resampling</td>
</tr>
<tr>
<td>train set size</td>
<td>20,900 examples in total</td>
</tr>
<tr>
<td>validation data</td>
<td></td>
</tr>
<tr>
<td>no. of positive examples</td>
<td>1,000 windows with letter ‘A’</td>
</tr>
<tr>
<td>no. of negative examples</td>
<td>10,000 windows with letters other than ‘A’ plus random samples of backgrounds on each stage after resampling</td>
</tr>
<tr>
<td>test set size</td>
<td>11,000 examples in total</td>
</tr>
<tr>
<td>test data</td>
<td></td>
</tr>
<tr>
<td>no. of positive examples</td>
<td>20,000 windows with letter ‘A’</td>
</tr>
<tr>
<td>no. of negative examples</td>
<td>110,000 windows with letters other than ‘A’ plus random samples of backgrounds</td>
</tr>
<tr>
<td>test set size</td>
<td>1,020,000 examples in total</td>
</tr>
</tbody>
</table>

Our goal was to detect targets (‘A’ letters) regardless of their rotation. In training images, the letters were allowed to rotate randomly within a limited range of ±45°. In test images, the letters were allowed to rotate randomly within the full range of 360°. As features, we applied 540 modules of Zernike moments (see [3] for details).

Fig. 3 presents examples of detection outcomes obtained by best detectors trained to satisfy 10⁻³ and 10⁻⁴ FAR requirements.

Table 5 presents a comparison of obtained cascades. Despite the small number of features (comparing to the previous experiment), the proposed methods still allow to reduce the expected number of features. FAR and sensitivity measures obtained on the validation set, satisfy the final requirements regardless of the training method. Accuracy measures observed on the test set are slightly worse. The best expected values of the number of feature were again obtained by the UGM-G approach: 2.5682 for $A = 10^{-3}$, and 3.7318 for $A = 10^{-4}$. UGM approach took second place.

5.4 Parallelization

We remarked that for clear interpretation of results, time measurements are reported for single threaded executions [ST], even though we normally apply multiple threads. Time gains that we report for the smallest expectations, may seem very small in ST mode. It should be explained that an improvement of e.g. a 10 ms [ST] constitutes a 6.5% difference per thread and allows for approximately 1 frame more. With 4 threads we observe approximately 46 ms–46 ms reductions, implying 2 FPS more. Common 8-threaded machines or GPUs with even more threads scale the gain further.

6 Conclusion

In our opinion, training a cascade of classifiers should be always carried out with the primary focus on the expected number of extracted features, because this quantity reflects directly how fast a cascade operates. In this paper we have demonstrated that the expected value can be improved by relaxation of per-stage requirements. This is achieved by taking advantage of the slack present between the worst-case constant requirements and the actual rates (detection / FAR).
Table 3. “Face detection” — cascades with $K = 5$ stages for different training approaches.

<table>
<thead>
<tr>
<th>Training algorithm</th>
<th>Cascade</th>
<th>Expected value</th>
<th>Validation sensitivity</th>
<th>FAR</th>
<th>Test sensitivity</th>
<th>avg(ν)</th>
<th>Detection time</th>
<th>window</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI</td>
<td>9</td>
<td>18</td>
<td>26 30 38</td>
<td>0.9403</td>
<td>0.9500</td>
<td>0.9572</td>
<td>16.21</td>
<td>89</td>
<td>0.680</td>
</tr>
<tr>
<td>UGM</td>
<td>9</td>
<td>17</td>
<td>32 29 29</td>
<td>0.9504</td>
<td>0.9500</td>
<td>0.9555</td>
<td>16.58</td>
<td>90</td>
<td>0.687</td>
</tr>
<tr>
<td>UGM-G</td>
<td>4</td>
<td>14</td>
<td>35 40 52</td>
<td>0.9550</td>
<td>0.9500</td>
<td>0.9660</td>
<td>50</td>
<td>88</td>
<td>0.675</td>
</tr>
</tbody>
</table>

Table 4. “Face detection” — cascades with $K = 10$ stages for different training approaches.

<table>
<thead>
<tr>
<th>Training algorithm</th>
<th>Cascade</th>
<th>Expected value</th>
<th>Validation sensitivity</th>
<th>FAR</th>
<th>Test sensitivity</th>
<th>avg(ν)</th>
<th>Detection time</th>
<th>window</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI</td>
<td>4</td>
<td>6</td>
<td>16 13 11 14 24 26 34</td>
<td>0.9403</td>
<td>0.9500</td>
<td>0.9552</td>
<td>13.91</td>
<td>73</td>
<td>0.561</td>
</tr>
<tr>
<td>UGM</td>
<td>6</td>
<td>14</td>
<td>13 10 13 18 18 17</td>
<td>0.9504</td>
<td>0.9500</td>
<td>0.9575</td>
<td>13.34</td>
<td>72</td>
<td>0.658</td>
</tr>
<tr>
<td>UGM-G</td>
<td>6</td>
<td>15</td>
<td>18 24 30 60 100</td>
<td>0.9550</td>
<td>0.9500</td>
<td>0.9522</td>
<td>15.11</td>
<td>67</td>
<td>0.512</td>
</tr>
</tbody>
</table>

Table 5. “Synthetic A letters” — cascades with $K = 5$ stages for different training approaches.

<table>
<thead>
<tr>
<th>Training algorithm</th>
<th>Cascade</th>
<th>Expected value</th>
<th>Validation sensitivity</th>
<th>FAR</th>
<th>Test sensitivity</th>
<th>avg(ν)</th>
<th>Detection time</th>
<th>window</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI</td>
<td>2</td>
<td>2.115, 0.2128, 0.1893, 0.2180, 0.2098</td>
<td>0.9540</td>
<td>0.9500</td>
<td>0.9313</td>
<td>2.56</td>
<td>127</td>
<td>6.77</td>
<td></td>
</tr>
<tr>
<td>UGM</td>
<td>2</td>
<td>0.2128, 0.1893, 0.2926, 0.3910</td>
<td>0.9500</td>
<td>0.9500</td>
<td>0.9244</td>
<td>2.50</td>
<td>125</td>
<td>6.67</td>
<td></td>
</tr>
<tr>
<td>UGM-G</td>
<td>2</td>
<td>0.2128, 0.2954, 0.2313, 0.2812</td>
<td>0.9500</td>
<td>0.9500</td>
<td>0.9351</td>
<td>2.50</td>
<td>124</td>
<td>6.61</td>
<td></td>
</tr>
</tbody>
</table>

Requirements: $D = 10$ windows per image: 130 971
observed during training. Both the mathematical considerations and practical experiments confirmed that this idea can lead to faster classifiers satisfying the same final requirements.

One of the plans for our future research is to incorporate the presented idea into a “branch-and-bound” search technique and to train cascades via both relaxation and searching.

ACKNOWLEDGEMENTS

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

REFERENCES


