Cascade training as a tree search with Dijkstra's algorithm

Dariusz Sychel[1][0000−0001−9835−869X], Aneta Bera[1][0000−0002−0456−9451], and Przemysław Klęsk[1][0000−0002−5579−187X]

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

Abstract. We propose a general algorithm that treats cascade training as a tree search process working according to *Dijkstra's algorithm* in contrast to our previous solution based on the *branch-and-bound* technique. The reason behind the algorithm change is reduction of training time. This change does not affect in anyway the quality of the final classifier. We conduct experiments on cascades trained to become face or letter detectors with Haar-like features or Zernike moments being the input information, respectively. We experiment with different tree sizes and different branching factors. Results confirm that training times of obtained cascades, especially for large heavily branched trees, were reduced. For small trees, the previous technique can sometimes achieve better results but the difference is negligible in most cases.

Keywords: Cascade of classifiers · Dijkstra's algorithm · Training Time Reduction · Tree search.

1 Introduction

Dijkstra's algorithm is a useful tool for finding the shortest paths between nodes in a graph. Many practical applications of this algorithm or its modifications can be found, e.g. airport automated guided vehicles (AGV) path optimization [16], evacuation route optimization under real-time toxic gas dispersion through computational fluid dynamics (CFD) simulation and Dijkstra's algorithm [15] or judgment of railway transportation path presented in [4].

Cascades of classifiers [13][14] were designed to work as classifying systems operating under two conditions: (1) very large number of incoming requests, (2) significant classe imbalance. A cascade should vary its computational effort 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. We remark that the optimization problem we try to solve in this research (and the previous one [9]) is to build such a cascade that minimize the *expected number of features* applied by the cascade (formal definition show in Section 2.3).

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63751-3_17

Despite the development of deep learning, recent literature shows that cascades of classifiers are still applied in detection systems or batch classification jobs e.g. GPGPU-based (General-Purpose Graphics Processing Unit) parallel version of eyes detecting cascade was used for driving an intelligent wheelchair [7]. Cascade with improved memory consumption was applied for remote sensing tasks [12]. Cascades of classifiers were also used for kitchen safety monitoring [2] or for object detection for robot cars [6]. Moreover authors of [1] show that classifier cascade can be significantly faster than YOLO (you only look once) classifier without substantial reduction of accuracy. Their comparison was conducted on the driver drowsiness detection problem.

In our previous work [10] we provided and proved a theoretical result demonstrating that the presence of slack between the constant per-stage requirements (on accuracy measures) used in the original cascade algorithm and actual rates observed while learning, allows to introduce new *relaxed* requirements for each successive stage and still complete the training procedure successfully. 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. Taking advantage of the relaxation, we proposed new stage-wise training algorithms that apply two approaches: uniform or greedy. They differ in the way the slack accumulated so far becomes "consumed" later on. Results obtained by the greedy algorithm (UGM-G) were better in most cases and this variant became the default one for further research.

This leads us to a new general algorithm that treats cascade training as a tree search process working according to the *branch-and-bound* technique [9]. Successive tree levels correspond to successive cascade stages. Sibling nodes represent variants of the same stage with different number of features applied. We provided suitable formulas for lower bounds on the expected value to be optimized. While searching, we observe suitable lower bounds on partial expectations and prune tree branches that cannot improve the best-so-far result. 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. Both exact and approximate variants of the approach were formulated in [9]. Our results confirmed shorter operating times of cascades obtained owing to the reduction in the number of extracted features. The main contribution of this paper is a new training algorithm for cascade training. It is performed via a tree search procedure that uses *Dijkstra's algorithm* in order to reduce the training time compared to the *branch-and-bound* technique that was used in the previous work. For our purposes we consider the *single-source single-destination* variant of Dijkstra's algorithm [5] (rather than the single-source *all* shortest paths). Additionally, the reduction of training time does not have any negative impact on the *expected number of features* in obtained final cascades. Both approaches the new and the old one (exact branch-and-bound version) result in exactly the same cascades. The new technique is recommended for large heavily branched trees. In case of small trees, the previous technique can sometimes achieve better results in terms of training time but the difference is in fact negligible in most cases.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63751-3_17

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 (false alarm rates) values on successive stages,
- $(d_1, d_2, \ldots, d_K)$ — sensitivities (detection rates) on successive stages,
- A — required FAR for the whole cascade,
- D — required detection rate (sensitivity) for the whole cascade,
- $a_{\max} = A^{1/K}$ — per-stage FAR requirement,
- $d_{\min} = D^{1/K}$ — per-stage sensitivity requirement,
- $F = (F_1, F_2, \ldots, F_K)$ — 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 ($D_k = \prod_{1 \leqslant i \leqslant k} d_i$),
- θ_k — decision threshold for classifier F_k, it should be set to the minimal value that satisfies $d_k \geqslant d_{\min}$,
- $(p, 1-p)$ — 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 with constant per-stage requirements can be presented with the pseudo-code below (Algorithm 1).

2.3 Expected number of extracted features

Cascade performance is directly dependent on the average number of features used per window regardless of the learning method, therefore there is a direct connection between the expected number of features and detection time. To support this claim Table 1 show impact of expected number of features on detection time for three example classifiers obtained in our previous work [10].

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63751-3_17

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 θ_{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 $F = (F_1, F_2, \ldots, F_k)$.

Table 1. "Face detection" — impact of expected number of features on detection time ($A = 0.001, D = 0.95$).

	Cascade	Expected value	FAR	Sensitivity	Detection time	
					image [ms]	window [μs]
n_k	9 16 21 22 39					
a_k	0.2468, 0.2548, 0.2234, 0.2635, 0.2606	14.7220	0.000964	0.9510	88	0.675
n_k	9 18 26 30 38					
a_k	0.2468, 0.2214, 0.2299, 0.2468, 0.2370	15.3571	0.000735	0.9520	89	0.680
n_k	9 17 32 29 29					
a_k	0.2468, 0.2516, 0.2450, 0.2303, 0.2798	15.7243	0.000980	0.9510	90	0.687

Definition-based formula A cascade stops operating after a certain number of stages. Therefore the possible outcomes of the random variable of interest, describing the disjoint events, are: n_1, $n_1 + n_2$, …, $n_1 + n_2 + \cdots + n_K$. 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.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63751-3_17

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

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

In practical applications the true probability distribution underlying the data is unknown. Since the probability p of the positive class is very small, the expected value can be accurately approximated using only the summands related to the negative class as follows:

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

In our previous works [10][9] we focused on creating an algorithm that tries to decrease this quantity compared to the original Viola and Jones algorithms. Because the solution proposed in this paper is a modification of [9], the expected value also plays a critical role in it, as is used to establish the priority of nodes in the priority queue used in Dijkstra's algorithm.

2.4 Relaxed per-stage requirements

Instead of constant per-stage requirements proposed in the original approach we continue to use the greedy variant of relaxed per-stage requirements, proposed by us in [10], since applying them results in cascades with lower expected number of features. As a reminder:

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

$$a_k = (1 - \epsilon_k) a_{max}, \qquad d_k = (1 + \delta_k) d_{min}, \tag{5}$$

where ϵ_k, δ_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:*

$$a_{k+1} \leqslant \frac{a_{max}}{(1 - \epsilon_{\leqslant k})^{1/(K-k)}}, \qquad d_{k+1} \geqslant \frac{d_{min}}{(1 + \delta_{\leqslant k})^{1/(K-k)}}, \tag{6}$$

or

$$a_{k+1} \leqslant \frac{a_{max}}{1 - \epsilon_{\leqslant k}}, \qquad d_{k+1} \geqslant \frac{d_{min}}{1 + \delta_{\leqslant k}}, \tag{7}$$

where $1 - \epsilon_{\leqslant k} = \prod_{1 \leqslant i \leqslant k} (1 - \epsilon_i)$ *and* $1 + \delta_{\leqslant k} = \prod_{1 \leqslant i \leqslant k} (1 + \delta_i)$.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63751-3_17

For proof see [10].

The greedy per-stage requirements (UGM-G) can be expressed in terms of A, D constants and a_i, d_i rates observed so far, that is for $i \leqslant k$, as follows:

$$a_{\max,k+1} = \frac{A^{\frac{k+1}{K}}}{\prod_{1\leqslant i\leqslant k} a_i}, \qquad d_{\min,k+1} = \frac{D^{\frac{k+1}{K}}}{\prod_{1\leqslant i\leqslant k} d_i}. \tag{8}$$

2.5 Cascade training as a tree search

When cascade training is treated 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 have an odd number of children nodes. They represent variants of a subsequent stage with slightly different number of features. The children are processed from left to right until the stop condition is met.

The size of the tree can 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 is equal to C, an odd number, e.g. $C = 5$ (mandatory middle node, $\frac{C-1}{2}$ nodes created by removing features from it, $\frac{C-1}{2}$ nodes created by adding new features). At deeper levels the branching factor is one.

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 (9)

$$\widehat{E}\Big((n_1,\ldots,n_{k+1})\Big) = \sum_{1\leqslant j\leqslant k} n_j \prod_{1\leqslant i<j} a_i + n_{k+1}\prod_{1\leqslant i<k+1} a_i = \widehat{E}\Big((n_1,\ldots,n_k)\Big) + n_{k+1}\prod_{1\leqslant i<k+1} a_i. \tag{9}$$

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[1]. In other words, pruning can be applied because formula (9) provides a lower bound on the final unknown expectation. Fig. 1 provides a symbolic illustration of a search tree with pruning.

[1] Initial value of $\widehat{E}^*$ is set to ∞, after first cascade satisfying (A, D) requirements finish its training, $\widehat{E}^*$ represents its expected number of features.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63751-3_17

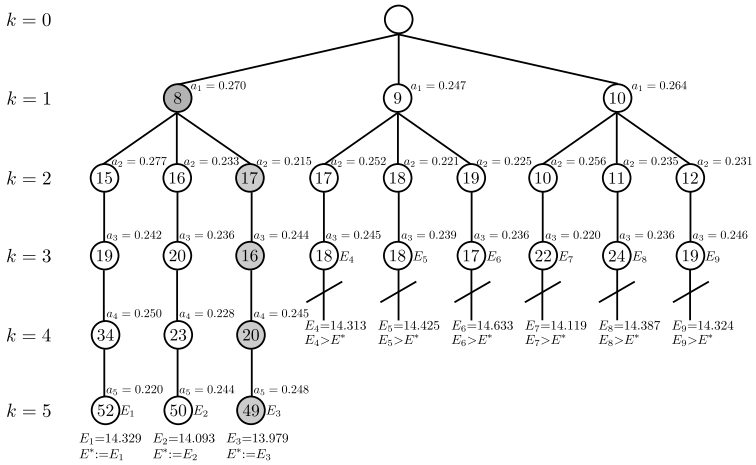

Fig. 1. Cascade training as a tree search with pruning — example illustration.

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 most recent 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 (9).
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^*).

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63751-3_17

The outermost recursion call (initial call) for Algorithm 2 is

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

The TrainStage function in Algorithm 2 correspond to a single execution of external while loop inside Algorithm 1. It results in a single ensemble trained using per-stage requirements. The requirements can be calculated as standard geometric means (classical VJ-style), leading to constant per-stage requirements for the whole training, or as updated geometric means (UGM).

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

As we have shown in [9] the training time can be reduced even more by the following approximate branch-and-bound approach. 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 (9) for stage $k+1$, but it is needed for stage $k+2$. Therefore, the only unknown preventing us from calculating the exact partial expectation for stage $k+2$ is n_{k+2}. We propose a formula that allows to approximate this value: $n_{k+2} \geqslant \alpha\, n_{k+1}$.

Dijkstra's algorithm (single-source single-destination) for cascade tree search

As can be noticed, the approach presented so far is based on a depth-first traversal accelerated by the branch-and-bound technique. One should realize that when the search process is conducted recursively from left to right, the training time can in some cases be significantly longer (e.g. if the solution is in the far right part of tree).

In order to overcome the above disadvantage, we postulate to replace the previous traversal order with the *Dijkstra's algorithm* in which the priority of each node is equal to its partial expected value on the number of features. This means that tree nodes with lower $\hat{E}$ values shall be visited sooner, and therefore the most promising nodes shall be always evaluated first regardless of their position the cascade search tree. The new training procedure is presented below as Algorithm 3.

Object *node* in Algorithm 3 represents a single tree node. Every *node* object contains the following fields: C — number of children created after the node's training is finished, k — tree level (equivalent to the cascade stage), A_k, D_k — FAR/sensitivity for the cascade ended at that node.

The way of creating new children nodes is the same as in Algorithm 2. Once the middle child is created, the left siblings are created by removing weak classifiers from the middle child, whereas the right siblings must undergo further training in order to add new weak classifiers (and features).

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63751-3_17

Algorithm 3 Training cascade of classifiers via tree search with Dijkstra's algorithm

procedure TRAINTREECASCADEDIJKSTRA($\mathcal{D}$, A, D, K, $\mathcal{V}$, C, L)
From $\mathcal{D}$ take subset $\mathcal{P}$ with all positive examples, and subset $\mathcal{N}$ with all negative examples.
Insert *root* (empty node with $\widehat{E} = 0$) into priority queue *open*.
while *open* not empty **do**
Take *node* with the smallest $\hat{E}$ from *open*.
node.$C := 1$.
if *node*.$k < L$ **then**
node.$C := C$.
if *node*.$A_k > A$ or *node*.$D_k < D$ **then**
if *node*.$k < K$ **then**
Train C *children* of current *node* according to TRAINTREECASCADE procedure.
for all $c \in$ *children* **do**
if *node*.$k + 1 < K$ **then**
Prepare new training set $\mathcal{D}_{k+1,c}$ and new validation set $\mathcal{V}_{k+1,c}$.
Calculate expectation $\widehat{E}$ for cascade $F \| F_{k+1,c}$ using (9).
Insert child c into priority queue *open*.
Insert *node* into list *closed*.
else if *node*.$A_k < A$ and *node*.$D_k > D$ **then**
return *node*

3 Experiments

Similar to our previous work [9][10] research was conducted on machine with Intel Core i7-4790K 4/8 cores/threads, 8MB cache. In all experiments we apply *RealBoost+bins* [8] as the main learning algorithm, producing ensembles of weak classifiers as successive cascade stages. Each weak classifier is based on a single selected feature. In letter "A" detection task we used computer fonts prepared by T.E. de Campos et al. [3]. In face detection task for training purpose we used faces cropped from 3000 images, looked up using Google Images search engine. Test set contains faces from Essex facial images collection [11]. More details about experimental setup can be found in mentioned articles.

3.1 Average time per node

As our previous work has shown, the further a weak classifier is in a cascade the more time it takes to train it. This is mainly due to the time needed to perform the resampling of the training and validation sets. With each weak classifier added to a cascade its FAR value decreases. Because of this, it becomes more and more difficult to find an image window misclassified as a positive.

Table 2 shows how the node average training time and the average resampling times at given stages looked like in experiments. Average times were calculated based on cascades trained to satisfy final requirement equal to $A = 0.001$ (FAR)

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63751-3_17

and $D = 0.95$ (sensitivity). The decrease in the node average training time for higher K values is associated with a smaller number of features per stage for more extensive cascades (that satisfy same final requirements), which can be observed in the Table 5. When it comes to resampling times, the average times shown in the Table 2 confirm a significant increase of time needed for successive stages.

Table 2. Average node training and resampling times per tree level (A = 0.001, D = 0.95)

	Avg. Training Time [s]		Avg. Resampling Time per Stage [s]								
Stages	per Stage	Total	1	2	3	4	5	6	7	8	9
HAAR-like features											
K = 5	280	1 400	36	332	2 011	7 286					
K = 10	164	1 639	6	34	106	271	537	913	1 916	4 553	8 087
Zernike moments											
K = 10	2	19	16	41	52	129	224	430	848	1 739	3 701

3.2 Training times

The conducted research shows that the proposed approach works especially well for large cascades. In the case of experiments with face detection (Table 3) we can see that for a high value of FAR $A = 0.01$ even for relatively small cascades with only 5 stages Dijkstra's algorithm allowed to reduce the training time, but for more demanding settings $A = 0.001$ the obtained results are similar or worse to the ones achieved by pruning technique with high α factor. With the increase of the tree size the effectiveness of Dijkstra's algorithm (in terms of training time reduction) rises significantly. It is also worth recalling at this point that the pruning algorithm proposed in [9] approximates the partial expected value, therefore setting a high value of α increases the risk of omitting the cascade with the lowest expected value. On the other hand, Dijkstra's algorithm guarantees finding the optimal solution (i.e. the cascade with the minimal expected value contained in the current tree).

In the case of the second group of experiments (Table 4) — detection of letter 'A', we can notice that the results are similar. Because only few Zernike moments per stage were needed in order to detect letter 'A' with $A = 0.001$, we could not test trees with higher values of C, L and K parameters. Instead we decided to train more demanding classifiers with $A = 0.0001$ in order to show the impact of the proposed solution.

The approach proposed in this paper works particularly well for large multi-level trees with a high number of nodes. In practice, this allows a larger combination of cascades to be searched, without significantly increasing the training time compared to the classical cascade learning approach.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63751-3_17

Table 3. Training time comparison (Face Detection — HAAR-like features)

Training algorithm	K	Splits (L)	Children per split (C)	Nodes	A	D	Trained Nodes DFS + pruning $\alpha = 0.0$	DFS + pruning $\alpha = 1.2$	Dijkstra open	Dijkstra close	Training time DFS + pruning $\alpha = 0.0$	DFS + pruning $\alpha = 1.2$	Dijkstra
Face Detection (HAAR-like features)													
UGM-G	5	1	3	15	0.01	0.95	13	13	11	9	3 130s	2 925s	2 564s
UGM-G	5	1	3	15	0.001	0.95	12	11	12	11	13 375s	11 624s	11 960s
UGM-G	5	1	5	25	0.001	0.95	16	13	17	13	12 238s	10 525s	16 543s
UGM-G	5	1	7	35	0.001	0.95	19	14	20	14	12 039s	10 061s	15 957s
UGM-G	5	2	3	39	0.001	0.95	30	25	27	22	43 642s	30 232s	96 319s
UGM-G	10	1	3	30	0.001	0.95	24	23	17	15	27 916s	25 929s	14 076s
UGM-G	10	1	5	50	0.001	0.95	40	36	25	21	50 963s	49 467s	20 075s
UGM-G	10	1	7	70	0.001	0.95	46	41	31	25	51 679s	50 406s	25 427s
UGM-G	10	2	3	84	0.001	0.95	56	48	31	23	98 053s	90 897s	23 975s
UGM-G	15	1	3	45	0.001	0.95	44	43	33	31	65 381s	58 799s	19 421s
UGM-G	15	1	5	75	0.001	0.95	65	61	47	43	94 344s	90 863s	25 231s
UGM-G	15	1	7	105	0.001	0.95	85	79	58	52	130 437s	121 311s	28 468s
UGM-G	15	2	3	129	0.001	0.95	86	78	60	53	122 668s	106 741s	31 116s

Table 4. Training time comparison (Letter 'A' Detection — Zernike moments)

Training algorithm	K	Splits (L)	Children per split (C)	Nodes	A	D	Trained Nodes DFS + pruning $\alpha = 0.0$	DFS + pruning $\alpha = 1.2$	Dijkstra open	Dijkstra close	Training time DFS + pruning $\alpha = 0.0$	DFS + pruning $\alpha = 1.2$	Dijkstra
Letter 'A' Detection (Zernike moments)													
UGM-G	10	1	3	30	0.001	0.95	21	21	14	12	14 575s	14 211s	5 349s
UGM-G	10	1	3	30	0.0001	0.95	18	12	18	17	50 054s	43 121s	47 961s
UGM-G	15	1	3	45	0.0001	0.95	32	31	20	18	196 150s	192 421s	52 912s

3.3 Prunning efficiency

The results presented in Table 5 show the number of nodes visited by each method. This table is an extension of a table from [9] with additional results from Dijkstra's algorithm. In the case of Dijkstra's algorithm, two new values were reported: *open* — means how many nodes were generated and added to the priority queue (nodes in the open queue were trained, but their descendants were not yet created and hence data resampling was not yet needed), *closed* — represents the number of nodes that have been fully processed. The values in the *closed* column are always non-greater than the values of *open* column. It should be remarked that closed nodes counts correspond to number of trained nodes reported for the previous algorithms (open nodes are of lighter computational costs). The difference between closed and open counts tells us how many children were created but never visited.

In this experiment we decided to use three α values: $\alpha = 0.0$ is equivalent to the exact pruning method, $\alpha = 0.8$ to pruning with low risk of missing the optimal solution, $\alpha = 1.2$ corresponds to a more aggressive pruning. Higher values of α were not used because of high risk of missing the optimal solution.

As we can see, the proposed algorithm, thanks to the use of a priority queue, allowed for a significant reduction in the number of nodes compared to the exact pruning method. This difference is caused by the fact that in case of the branch-and-bound method the effectiveness of the approach depends on the order in which children nodes are visited (similar to other branch-and-bound based methods its computational complexity is $O(C^K)$, in optimistic case it is $\Omega(C^{\frac{1}{2}K})$, average complexity is $\Theta(C^{\frac{3}{4}K})$), while in case of Dijkstra's method we

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63751-3_17

directly compare expected number of features of all reachable nodes. In most cases the number of nodes visited was also smaller than the one obtained from the greedy pruning with $\alpha = 1.2$.

Table 5. Face detection (Haar-like features) — comparison of search tree-based approaches for different values of parameter C and L.

Training algorithm	Cascade	E(n)	Validation FAR	Validation sensitivity	Trained Nodes DFS + pruning exact	DFS + pruning approximate α: 0.0	DFS + pruning approximate α: 0.8	DFS + pruning approximate α: 1.2	Dijkstra open	Dijkstra close
			Requirement: 10^{-3} = **0.001**	**0.95**						
VJ	9 18 26 30 38 0.2468, 0.2214, 0.2299, 0.2468, 0.2370	15.36	0.00074	0.9520						
TREE-C3-L1 VJ	8 16 22 35 55 0.2703, 0.2329, 0.2188, 0.2419, 0.2487	14.37	0.00083	0.9520	12/15	11/15	10/15	12	11	
TREE-C3-L1 UGM	8 16 22 27 37 0.2703, 0.2329, 0.2188, 0.2601, 0.2439	14.21	0.00087	0.9510	12/15	11/15	10/15	12	11	
TREE-C3-L1 UGM-G	8 16 22 25 40 0.2703, 0.2329, 0.2188, 0.2713, 0.2646	14.20	0.00099	0.9510	12/15	11/15	11/15	12	11	
TREE-C3-L2 VJ	8 16 22 35 55 0.2703, 0.2329, 0.2188, 0.2419, 0.2487	14.37	0.00083	0.9520	30/39	26/39	23/39	27	23	
TREE-C3-L2 UGM	8 16 22 27 37 0.2703, 0.2329, 0.2188, 0.2601, 0.2439	14.21	0.00087	0.9510	29/39	25/39	24/39	28	21	
TREE-C3-L2 UGM-G	8 16 22 25 40 0.2703, 0.2329, 0.2188, 0.2713, 0.2646	14.20	0.00099	0.9510	30/39	26/39	25/39	27	22	
TREE-C5-L1 VJ	7 18 23 30 40 0.2770, 0.2134, 0.2500, 0.2426, 0.2408	13.93	0.00086	0.9520	17/25	15/25	15/25	18	14	
TREE-C5-L1 UGM	7 18 21 26 43 0.2770, 0.2134, 0.2535, 0.2577, 0.2456	13.78	0.00095	0.9510	17/25	15/25	14/25	17	14	
TREE-C5-L1 UGM-G	7 18 17 33 30 0.2770, 0.2134, 0.2648, 0.2514, 0.2452	13.62	0.00097	0.9510	16/25	15/25	13/25	17	13	
VJ	4 6 16 13 11 17 14 21 24 45 0.45, 0.45, 0.48, 0.42, 0.50, 0.47, 0.45, 0.49, 0.50, 0.47	12.37	0.00051	0.9560						
TREE-C3-L1 VJ	4 6 16 13 11 17 14 21 24 45 0.45, 0.45, 0.48, 0.42, 0.50, 0.47, 0.45, 0.49, 0.50, 0.47	12.37	0.00051	0.9560	24/30	23/30	22/30	20	18	
TREE-C3-L1 UGM	3 6 5 10 11 14 24 15 15 18 0.76, 0.34, 0.45, 0.50, 0.48, 0.45, 0.52, 0.50, 0.54, 0.51	11.51	0.00088	0.9510	19/30	17/30	17/30	20	18	
TREE-C3-L1 UGM-G	4 6 5 14 8 12 17 16 14 34 0.45, 0.44, 0.59, 0.51, 0.49, 0.52, 0.50, 0.47, 0.54, 0.45	10.70	0.00090	0.9510	24/30	23/30	23/30	17	15	
TREE-C3-L2 VJ	3 6 5 10 11 15 17 30 40 19 0.76, 0.34, 0.45, 0.50, 0.48, 0.45, 0.46, 0.48, 0.49, 0.50	11.59	0.00068	0.9550	49/84	44/84	42/84	42	34	
TREE-C3-L2 UGM	3 6 5 10 11 14 24 15 15 18 0.76, 0.34, 0.45, 0.50, 0.48, 0.45, 0.52, 0.50, 0.54, 0.51	11.51	0.00088	0.9510	44/84	39/84	35/84	39	31	
TREE-C3-L2 UGM-G	4 6 5 14 8 12 17 16 14 34 0.45, 0.44, 0.59, 0.51, 0.49, 0.52, 0.50, 0.47, 0.54, 0.45	10.70	0.00090	0.9510	56/84	55/84	48/84	31	23	
TREE-C5-L1 VJ	4 6 16 13 11 17 14 21 24 45 0.45, 0.45, 0.48, 0.42, 0.50, 0.47, 0.45, 0.49, 0.50, 0.47	12.37	0.00051	0.9560	38/50	37/50	36/50	30	26	
TREE-C5-L1 UGM	3 6 5 10 11 14 24 15 15 18 0.76, 0.34, 0.45, 0.50, 0.48, 0.45, 0.52, 0.50, 0.54, 0.51	11.51	0.00088	0.9510	33/50	30/50	29/50	29	25	
TREE-C5-L1 UGM-G	4 6 5 14 8 12 17 16 14 34 0.45, 0.44, 0.59, 0.51, 0.49, 0.52, 0.50, 0.47, 0.54, 0.45	10.70	0.00090	0.9510	40/50	39/50	36/50	25	21	

3.4 Detection examples

Figure 2 presents examples of face detection obtained by classifiers with the lowest expected number of extracted features trained by tree search procedure with FAR values set to $A = 10^{-3}$ and $A = 10^{-4}$ respectively. Sensitivity for both classifiers was set to $D = 0.95$. The decision threshold for classifiers was set to 1.0.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63751-3_17

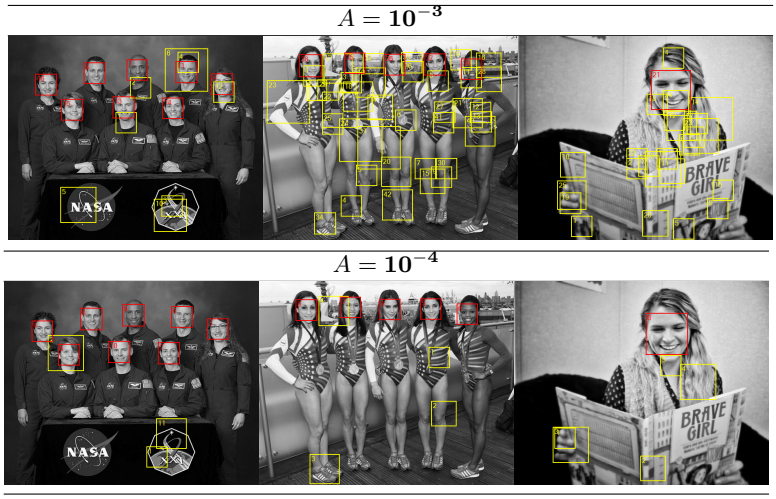

Fig. 2. "Face detection": detection examples. False alarms marked in yellow.

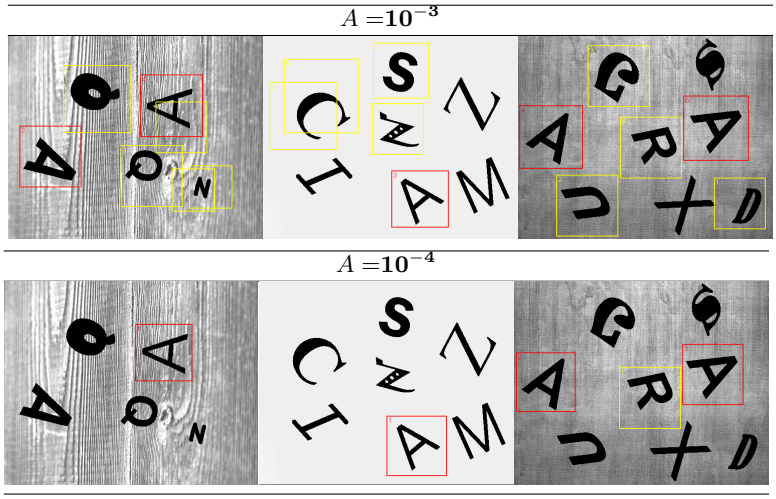

Fig. 3. "Synthetic A letters": detection examples.

Similarly, Figure 3 shows examples of letter 'A' detection obtained by classifiers with the lowest expected number of extracted features trained by tree search procedure with FAR values set to $A = 10^{-3}$ and $A = 10^{-4}$ respectively. Sensitivity for both classifiers was also set to $D = 0.95$. The decision threshold for classifiers was set to 0.0.

It should be recalled that the cascades obtained by both approaches (Dijkstra's algorithm and branch-and-bound method) are identical, the difference is in the time needed to find solution.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63751-3_17

4 Conclusions

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. In our previous research we propose to use a tree search-based training that allows to 'track' more than one variant of a cascade. This approach can be computationally expensive, but we have managed to reduce it with suitable branch-and-bound techniques.

The use of Dijkstra's algorithm for a tree search allows for further reduction of training times, especially in the case of complex trees with a large number of nodes. In addition, the aforementioned algorithm guarantees that the final cascade (returned as an outcome) always has the lowest expected value among all cascades contained in a given tree for the imposed settings of branching factor and tree depth (since priority queue in our approach sets nodes priority based on expected number of features in associated cascade and by taking into account that adding new stage to cascade can only increase expected value we can notice that after finding first cascade that satisfy requirements there is no possibility of finding other cascade that will improve the current expected value). This property is not necessarily satisfied in our previous method that use the approximate pruning.

References

1. Andrean, M.N., Shidik, G.F., Naufal, M., Al Zami, F., Winarno, S., Al Azies, H., Putra, P.L.W.E.: Comparing haar cascade and yoloface for region of interest classification in drowsiness detection. JURNAL MEDIA INFORMATIKA BUDIDARMA **8**(1), 272–281 (2024)
2. Bernabe, J.A., Dylan O. Catapang, J., Valiente, L.D.: Application of haar cascade classifier for kitchen safety monitoring. In: 2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS). vol. 1, pp. 343–348 (2023)
3. de Campos, T.E., et al.: Character recognition in natural images. In: Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal. pp. 273–280 (2009)
4. Di, J., Gao, R.: Research on railway transportation route based on dijkstra algorithm. In: 2021 International Conference on Big Data Analytics for Cyber-Physical System in Smart City. pp. 255–260. Springer Singapore (2022)
5. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**(1), 269–271 (1959)
6. Gharge, S., Patil, A., Patel, S., Shetty, V., Mundhada, N.: Real-time object detection using haar cascade classifier for robot cars. In: 2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC). pp. 64–70 (2023)
7. Ghorbel, A., Ben Amor, N., Abid, M.: Gpgpu-based parallel computing of viola and jones eyes detection algorithm to drive an intelligent wheelchair. Journal of Signal Processing Systems (2022)

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63751-3_17

8. Rasolzadeh, B., et al.: Response Binning: Improved Weak Classifiers for Boosting. In: IEEE Intelligent Vehicles Symposium. pp. 344–349 (2006)
9. Sychel, D., Klęsk, P., Bera, A.: Branch-and-Bound Search for Training Cascades of Classifiers. In: Computational Science – ICCS 2020. Springer International Publishing (2020)
10. Sychel, D., Klęsk, P., Bera, A.: Relaxed Per-Stage Requirements for Training Cascades of Classifiers. In: Frontiers in Artificial Intelligence and Applications – ECAI 2020. vol. 325, pp. 1523–1530. IOS Press (2020)
11. University of Essex: Face Recognition Data. https://cswww.essex.ac.uk/mv/allfaces/faces96.html (1997), [Online; accessed 11-May-2019]
12. Usilin, S.A., Slavin, O.A., Arlazarov, V.V.: Memory consumption and computation efficiency improvements of viola–jones object detection method for remote sensing applications. Pattern Recognition and Image Analysis **31**(3), 571–579 (2021)
13. Viola, P., Jones, M.: Rapid Object Detection using a Boosted Cascade of Simple Features. In: Conference on Computer Vision and Pattern Recognition (CVPR'2001). pp. 511–518. IEEE (2001)
14. Viola, P., Jones, M.: Robust Real-time Face Detection. International Journal of Computer Vision **57**(2), 137–154 (2004)
15. Wang, J., Yu, X., Zong, R., Lu, S.: Evacuation route optimization under real-time toxic gas dispersion through cfd simulation and dijkstra algorithm. Journal of Loss Prevention in the Process Industries **76** (2022)
16. Zhou, Y., Huang, N.: Airport agv path optimization model based on ant colony algorithm to optimize dijkstra algorithm in urban systems. Sustainable Computing: Informatics and Systems **35**, 100716 (2022)

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63751-3_17