Notes on Expected Computational Cost of Classifiers Cascade: A Geometric View

Dariusz Sychel, Przemysław Klęsk and Aneta Bera
Faculty of Computer Science and Information Technology, West Pomeranian University of Technology, ul. Żołnierska 49, 71-210 Szczecin, Poland

Keywords: Cascade of Clasifier, Detection, Expected Number of Features.

Abstract: A cascade of classifiers, working within a detection procedure, extracts and uses different number of features depending on the window under analysis. Windows with background regions can be typically recognized as negative with just a few features, whereas windows with target objects (or resembling them) might require thousands of features. The central point of attention for this paper is a quantity that describes the average computational cost of an operating cascade, namely — the *expected* value of the *number of features* the cascade uses. This quantity can be calculated explicitly knowing the probability distribution underlying the data and the properties of a particular cascade (detection and false alarm rates of its stages), or it can be accurately estimated knowing just the latter. We show three purely geometric examples that demonstrate how training a cascade with sensitivity / FAR constraints imposed per each stage can lead to non-optimality in terms of the computational cost. We do not propose a particular algorithm to overcome the pitfalls of stage-wise training, instead, we sketch an intuition showing that non-greedy approaches can improve the resulting cascades.

1 INTRODUCTION

The idea of classifiers cascade was originally presented by Viola and Jones in (Viola and Jones, 2001). The main motivation behind that idea is an observation that negative windows constitute a vast majority of all windows during a detection procedure. Regardless of the task (face detection, pedestrian detection, etc.) it is roughly fair to say that the negatives are present in at least 99.99% of all windows. For example in face detection, several face windows are typically detected among the total of 10^5 or even 10^6 windows, depending on the settings (image resolution, number of scans, detection window sizes and shifts). Therefore, the classifier working as the detector should *vary* the computational efforts applied to particular windows. Windows with background regions or obvious non-targets can be rejected as negatives based on the information from just a few features. On the other hand, more promising windows (with target objects or resembling such) can be analyzed gradually using more and more features, even up to thousands of features. The above operation process is achieved by dividing the overall detector into a certain number of stages. The features are then extracted in increasing portions, stage after stage. If some stage returns a negative response then the further calculations are suppressed and the analyzed window becomes classified as a negative. If all stages are passed through with a positive response then the analyzed window becomes classified as a positive.

Originally, Viola and Jones (Viola and Jones, 2001; Viola and Jones, 2004) proposed to use Haar-like features, loosely connected to Haar wavelets (Papageorgiou et al., 1998), within a cascade. This was done also for computational reasons, because the extraction of Haar-like features can be easily speeded up by integral images (Crow, 1984). Each stage of Viola and Jones' cascade was a strong classifier (an ensemble of week classifiers) trained by the AdaBoost algorithm (Friedman et al., 2000; Freund and Schapire, 1996). The weak classifiers were decision stumps based on single features yielding the smallest classification errors. Successive strong classifiers included into the cascade were trained on a data subset that was classified as positive in the previous stage. Thereby, each subsequent classifier faces a tougher task, since it has to deal with less obvious data examples (lying closer to the decision boundary), and therefore requires more features.

As Viola and Jones noted themselves, training a cascade is a difficult combinatorial optimization problem which involves the following crucial parameters: number of cascade stages, number of features per stage and selection of those features, decision thresholds per stage. It is worth to remark that this problem has not been ultimately solved yet. The way Viola and Jones decided to tackle it was by imposing the *final accuracy requirements* the whole cascade should meet in order to be accepted by the user. These requirements are defined by a pair of numbers: the wanted minimum detection rate (sensitivity) and the allowed maximum false alarm rate (FAR). Due to the probabilistic properties of the cascade structure, one can apply the geometric mean and translate the final requirements onto another pair of numbers: the *stage requirements*. If each stage satisfies such requirements then the whole cascade also satisfies the final requirements.

Many improvements to cascade training has been introduced over the years in the literature (Gama and Brazdil, 2000; Bourdev and Brandt, 2005; Pham and Cham, 2007; Li and Zhang, 2013; Vallez et al., 2015), most of them trying out different feature selection algorithms or subsampling methods. One should remark however that in many of these works the main optimization process is still guided by the stage requirements for the predefined number of stages. Also, a certain disorientation in the research arises due to unclear optimization criterions. One may wonder whether a cascade should: (1) maximize accuracy, (2) maximize AUC, (3) maximize sensitivity while satisfying a certain FAR, (4) minimize FAR while satisfying a certain sensitivity, (5) minimize its training time, (6) minimize the expected number of features used per window during detection, or perhaps still something else. Very often, the proposed algorithms are driven by some mixture of these criterions based on suitably formulated Lagrangians (Saberian and Vasconcelos, 2014; Shen et al., 2010; Shen et al., 2013).

It can be noted that 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 value of features and detection time.

1.1 Motivation

In this paper we concentrate on the *expected number of features* used by a cascade. We do not attempt at providing a new training algorithm that shall optimize this quantity directly. Instead, we intend to give the reader a geometric intuition on what pitfalls the stage-wise training can fall into. We show three geometric examples (“cuboid in the corner”, “3D cube trap”, “chessboard trap”) which demonstrate optimal cascades in terms of the above expectation and suboptimal ones resulting from the stage-wise training. Although the examples are arranged ‘manually’, they may reflect situations (or variations) present in real data sets.

1.2 Notation

Throughout the paper we use the following notation:

- K — number of cascade stages
- $(n_1, n_2, \ldots, n_K)$ — number of features used on each stage,
- $(d_1, d_2, \ldots, d_K)$ — sensitivities per stage (detection rates),
- $(a_1, a_2, \ldots, a_K)$ — FAR values per stage (alarms),
- p — true probability of the positive class (unknown in practice),
- $1-p$ — true probability of the negative class (unknown in practice),
- D — required sensitivity (for entire cascade),
- A — required FAR (for entire cascade),
- $d_{\min} = D^{1/K}$ — sensitivity required per stage,
- $a_{\max} = A^{1/K}$ — FAR required per stage.

For clarity, we explain that given a data pair $(\mathbf{x}, y)$ where $\mathbf{x}$ is the vector of features and $y \in \{-,+\}$ is the corresponding class label, the probabilistic meaning of d_k and a_k is as follows:

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

$$a_k = 1 - P\left(F_k(\mathbf{x}) = - \,|\, y = -, F_1(\mathbf{x}) = \cdots = F_{k-1}(\mathbf{x}) = -\right), \tag{2}$$

where $F_k(\mathbf{x})$ denotes the response of k-th stage.

2 EXPECTED NUMBER OF FEATURES

In this section we give two exact approaches and one approximate approach allowing to calculate the expected number of features used by a cascade.

2.1 Expected Value — Definition-based Approach

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:

$$E(n) = \sum_{k=1}^{K}\left(\sum_{i=1}^{k} n_i\right)\left(p\left(\prod_{i=1}^{k-1} d_i\right)\cdot(1-d_k)^{[k<K]} + (1-p)\left(\prod_{i=1}^{k-1} a_i\right)\cdot(1-a_k)^{[k<K]}\right) \quad (3)$$

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

2.2 Expected Value — Incremental Approach

By grouping the terms in (3) according to n_k the following alternative formula can be derived:

$$E(n) = \sum_{k=1}^{K} n_k\left(p\prod_{i=1}^{k-1} d_i \cdot + (1-p)\prod_{i=1}^{k-1} a_i\right) \quad (4)$$

From now on we shall be using expression (4), being simpler than (3).

2.3 Remarks

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:

$$E(n) \approx \sum_{k=1}^{K} n_k \prod_{i=1}^{k-1} a_i. \quad (5)$$

It is also interesting to remark that in the original paper (Viola and Jones, 2004) the authors propose an incorrect formula to estimate the expected number of features, namely:

$$E_{\mathrm{VJ}}(n) = \sum_{k=1}^{K} n_k \prod_{i=1}^{k-1} r_i, \quad (6)$$

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

$$E_{\mathrm{VJ}}(n) = \sum_{k=1}^{K} n_k \prod_{i=1}^{k-1} (pd_i + (1-p)a_i). \quad (7)$$

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 the same data point $\mathbf{x}$ does not change its class label while traveling along the cascade.

3 EXAMPLE 1: "CUBOID IN THE CORNER"

Consider a data distribution contained in the n-dimensional unit cube with the positive class, being a cuboid, placed in one of the cube's corners. The probability distribution over the entire problem domain $(x_1,\ldots,x_n) \in [0,1]^n$ is uniform. The positive class is represented by the set:

$$P = \{(x_1,\ldots,x_n) \in [0,1]^n : 0 \le x_i \le w_i\},$$

where $0 < w_1 \le w_2 \le \ldots \le w_n \le 1$ are widhts of the cuboid (set up non-decreasingly). The negative class is the complement:

$$N = [0,1]^n \setminus P.$$

The problem is deterministic — the classes can be separated unambiguously.

For our mathematical purposes we shall now consider an idealistic variant of the cascade training algorithm that works directly on the continuous distribution rather than on a finite data set.

Suppose the number of stages K and the final requirements D, A (sensitivity, FAR) for the entire cascade have been imposed. Each stage of the cascade shall be formed by a group of splits orthogonal to the axes (similar to decision stumps). The intersection implied by the splits shall indicate the region of positive response for the given stage. Splits are added successively, using one feature at a time. The following pseudocode serves as a sketch of the above procedure.

Algorithm 1: Viola Jones-style cascade training based on stage-wise sensitivity / FAR requirements (for the purposes of Example 1).

```
procedure TRAINCASCADEVJSTYLE($D$, $A$, $K$)
    $d_{\min} := D^{1/K}$, $a_{\max} := A^{1/K}$,
    for $k := 1,\ldots,K$ do
        $n_k := 0$, $d_k := 0$, $a_k := 1$
        while ($d_k < d_{\min}$ or $a_k > a_{\max}$) do
            find feature $x_i$ and split position $s_i$
                (along $x_i$) that minimize classifica-
                tion error within the remaining data
                region
            add the split $(x_i, s_i)$ to the current stage
            $n_k := n_k + 1$
            calculate $d_k$, $a_k$
        end while
    end for
end procedure
```

Let us consider the case of $D = 1$ (hence also $d_{\min} = 1$). It is possible to see that this condition implies the following two consequences:

1. splits must be taken at positions s_i exactly equal to w_i,
2. successive selected features shall correspond to their natural ordering $x_1, x_2, \ldots$.

The reason behind the first consequence is that for $s_i < w_i$ the 100% sensitivity cannot be achieved, whereas for $s_i > w_i$ the FAR shall be increased unnecessarily. The reason behind the second consequence is that widths of the positive cuboid have been set up in non-decreasing order ($w_1 \leq w_2 \leq \ldots \leq w_n$). Therefore, a split on x_i taken before a split on x_j with $i > j$ would result in a greater FAR (and thereby would not minimize the error). Fig. 1 illustrates the data distribution and optimal splits for the Example 1 and $n = 3$.

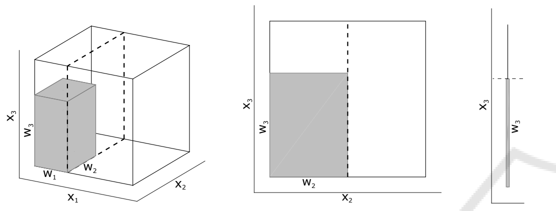

Figure 1: Example 1 “cuboid in the corner”: illustration for $n = 3$.

3.1 FAR Formula for “cuboid in the corner”

For further analysis of Example 1 we provide a useful formula for the false alarm rate. It assumes the optimal $s_i = w_i$ and expresses the FAR resulting from m new splits (i.e. using of m new features) provided that m_0 splits have already been done so far. Therefore the number m_0 can be treated as an offset value.

$$\mathrm{FAR}(m; m_0) = \frac{\left(\prod_{k=1+m_0}^{m_0+m} w_k\right) \cdot \left(1 - \prod_{k=1+m_0+m}^{n} w_k\right)}{1 - \prod_{k=1+m_0}^{n} w_k} \tag{8}$$

For example, if $n = 3$ and 1 initial split is to be made the resulting FAR becomes equal to:

$$\mathrm{FAR}(1;0) = \frac{w_1 (1 - w_2 \cdot w_3)}{1 - w_1 \cdot w_2 \cdot w_3}$$

Then, once the first split becomes fixed, and e.g. two new splits are to made, the FAR becomes:

$$\mathrm{FAR}(2;1) = \frac{w_2 \cdot w_3 (1 - 1)}{1 - w_2 \cdot w_3} = 0.$$

The $1 - 1$ value appearing in the numerator results from the empty product $\prod_{k=1+1+2}^{3} w_k$ equal to 1 by the definition.

Formula (8) lets us note again that features for the splits must be selected in their natural order. If the algorithm used some different order then it is easy to see that both factors in the numerator of (8) would increase while the denominator remains constant.

We remark that the following property is satisfied by the formula (8) for any natural numbers m_0, m_1, m_2:

$$\mathrm{FAR}(m_1 + m_2; m_0) = \mathrm{FAR}(m_1; m_0) \cdot \mathrm{FAR}(m_2; m_0 + m_1). \tag{9}$$

It means that the FAR value can be calculated ‘in portions’ and can be interpreted as follows. Taking $m_1 + m_2$ new splits after m_0 splits have been made is equivalent to taking first m_1 splits and then m_2 new splits (once the $m_0 + m_1$ former splits are fixed).

3.2 Optimal Cascades for “cuboid in the corner”

We are now going to look for optimal cascades for our “cuboid in the corner” example, imposing different K values. Two approaches shall be presented. In the first approach, exact solutions will be shown using some combinatorics and an exhaustive search. In the second, approximate solutions will be provided by a numerical technique.

Let n^* be the optimal number of features sufficient to achieve the imposed FAR value A.

$$n^* = \min\{m : \mathrm{FAR}(m; 0) \leq A\}$$

We note that it is possible that Algorithm 1, training the cascade, may find

$$N = n_1 + \cdots n_K > n^*$$

features. The overhead can be caused by the fact that all the stages must achieve the $(d_{\min}, a_{\max})$ requirements.

3.2.1 Exhaustive Combinatorial Search

First of all it is important to realize that the number of possible different cascades is

$$\binom{n^* - 1}{K - 1}. \tag{10}$$

We provide two explanations for this fact.

Explanation 1: Each stage must contain at least one weak classifier. Cascades ending with zeros in the sequence $(n_1, \ldots, n_k)$ can be omitted from considerations. Such cascade would have a larger $E(n)$ value than the cascades not ending with zeros but still using n^* features. Formula (10) can be interpreted as the number of ways one can place stage separators within a string $2, 3, \ldots, n^*$ that represents feature indexes (note that the first features is obligatory reserved for the first stage). For example, if $n^* = 7$ and $K = 3$ then a possible cascade one can build is e.g.:

$$\underline{2}, 3, \underline{4}, 5, 6, 7 \to 1|2, 3|4, 5, 6, 7 \to \begin{matrix} (n_1, & n_2, & n_3) \\ (1, & 2, & 4) \end{matrix}.$$

Explanation 2: Consider a mapping from the set of weak classifiers (for particular features) to the set of cascade stages:

$$\{1, 2, \ldots, n^*\} \to \{1, 2, \ldots, K\}$$

If the restriction that each stage must have at least one classifier is omitted for a moment then the number of possible mappings is equal to number of n^*-element combinations with repetitions of the set of K elements: $\binom{n^*+K-1}{n^*}$. The fact of taking into account the mentioned restriction can be represented by substituting $n^* := n^* - K$ (since K classifiers, 1 per stage, are fixed in the cascade from the start). This leads to the following number of cascades

$$\binom{n^* - K + K - 1}{n^* - K} = \binom{n^* - 1}{n^* - K} = \binom{n^* - 1}{K - 1},$$

which is equal to (10). The coding of the example cascade $(n_1, n_2, n_3) = (1, 2, 4)$ from the previous explanation can be now represented by the following string representing a combination with repetitions $*|**|****$.

Algorithm 2: Exhaustive combinatorial search for optimal cascade (for the purposes of Example 1).

```
procedure ExhaustiveCascades($D$, $A$, $K$, $N$)
    $E^* := \infty$                          ▷ best expectation so far
    $(n_1^*, \ldots, n_K^*) :=$ null         ▷ best cascade so far
    $c := (1, \ldots, N - K)$                ▷ initial combination
    while true do
        decode $c$ onto $(n_1, \ldots, n_K)$
        calculate $(d_1, \ldots, d_K)$ and $(a_1, \ldots, a_K)$
        if $\prod_{k=1}^{K} d_k \geq D$ and $\prod_{k=1}^{K} a_k \leq A$ then
            calculate $E$ according to (5)
            if $E < E^*$ then
                $E^* := E$
                $(n_1^*, \ldots, n_K^*) := (n_1, \ldots, n_K)$
            end if
        end if
        $c :=$NextCombination($c$, $N - 1$)
    end while
    return $(n_1^*, \ldots, n_K^*)$, $E^*$
end procedure
```

The NextCombination$(\cdot)$ represents a combinatorial iterating routine available in most programming environments[1]. The initial combination (without repetitions) becomes decoded onto the cascade $(N - K + 1, 1, 1, \ldots, 1)$, the next combination onto $(N - K, 2, 1, \ldots, 1)$, and the loop continues until the last cascade $(1, 1, 1, \ldots, N - K + 1)$.

Below, we report the results of experiments pertaining to Example 1 for the number of dimensions (features): $n = 30$ and $n = 50$. In both cases we set up the following arithmetic progression for the widths of positive cuboid in the corner: $w_1 = 0.8$, $\Delta = (1 - w_1)/n$ and $w_k = w_1 + (k - 1)\Delta$.

Experiments for *n* = *30*: The results presented in Table 1 compare the cascades obtained via Algorithm 1 (VJ-style cascade training) against the cascades from Algorithm 2 (combinatorial exhaustive search). The implementations were done in Wolfram Mathematica 10.4[2]. While calculating false alarm rates a_k, formula (8) was applied in both implementations. The total of all features in a cascade — $N = n_1 + \cdots + n_K$, obtained from the first algorithm, was used as an input to the second algorithm. We remark that the probability of the positive class in this experiment is $p = \prod_{k=1}^{n} w_k \approx 0.035633$. This probability together with (d_k) and (a_k) sequences allowed to calculate exact expected values $E(n)$.

As one can note all cascades obtained by the VJ-style algorithm (guided by stage requirements) have worse expected values $E(n)$ than the cascades obtained by the exhaustive combinatorial search. The latter cascades use the same number of features (denoted by N), but distribute them differently. The next table (Table 2) illustrates the fact that in two cases the number N can be still slightly decreased to N^* and the final requirements are still satisfied by cascades found via exhaustive search. Obviously, $E(n)$ values are simultaneously improved as well.

Experiments for *n* = *50*: Due to $n = 50$, the probability of the positive class for this experiment is $p = \prod_{k=1}^{n} w_k \approx 0.00415707$. As in the previous experiment we report two analogical tables. Table 3 confronts VJ-style cascades against the optimal ones (using the same totals N). Table 4 indicates improved totals N^* and the corresponding optimal cascades.

3.2.2 Approximate Numerical Technique

The difficulty in direct optimization of the expected value $E(n)$ lies in two facts: (i) the total of features at

[1]In our implementation we apply the `NextKSubset[·]` procedure from Wolfram Mathematica 10.4.

[2]Time results are reported for Intel Xeon CPU E3-1505M 2.8 GHz, 4 cores / 8 threads.

Table 1: Example 1 “cuboid in the corner” ($n = 30$): VJ-style cascades vs optimal cascades found by exhaustive search.

			resulting cascade			optimal cascade			
no.	K	A	(VJ-style algorithm)	N	$E(n)$	(exhaustive search)	$E(n)$	no. of cascades	search time [s]
1	2	10^{-2}	$(12,11)$	23	13.2884	$(7,16)$	10.9849	22	0.016
2	2	10^{-3}	$(18,10)$	28	18.6085	$(8,20)$	12.2173	27	0.016
3	3	10^{-2}	$(8,9,7)$	24	10.3641	$(4,6,14)$	8.74884	253	0.094
4	3	10^{-3}	$(12,11,5)$	28	13.5038	$(4,7,17)$	9.29271	351	0.141
5	4	10^{-2}	$(6,6,6,5)$	25	8.78596	$(3,4,6,10)$	7.62453	1540	0.516
6	4	10^{-3}	$(9,9,7,4)$	29	11.2031	$(3,4,7,15)$	8.21372	3276	1.188

Table 2: Example 1 “cuboid in the corner” ($n = 30$): improved totals of features N^*.

			optimal cascade			
no.	N	N^*	(exhaustive search, using N^*	$E(n)$	no. of cascades	search time [s]
3	24	23	$(4,6,13)$	8.59409	231	0.078
6	29	28	$(3,4,7,14)$	8.12233	2925	1.016

Table 3: Example 1 “cuboid in the corner” ($n = 50$): VJ-style cascades vs optimal cascades found by exhaustive search.

			resulting cascade			optimal cascade			
no.	K	A	(VJ-style algorithm)	N	$E(n)$	(exhaustive search)	$E(n)$	no. of cascades	search time [s]
1	5	10^{-2}	$(5,5,6,6,7)$	29	7.98656	$(2,3,4,7,13)$	7.08589	20475	7.281
2	5	10^{-3}	$(7,8,9,10,8)$	42	9.62653	$(2,3,5,9,23)$	7.61736	101270	44.766
3	6	10^{-2}	$(4,4,5,5,6,6)$	30	7.38845	$(2,2,3,4,7,12)$	6.72574	118755	44.453
4	6	10^{-3}	$(6,7,8,8,8,6)$	43	8.94712	$(2,3,3,5,9,21)$	7.12384	850668	387.328

Table 4: Example 1 “cuboid in the corner” ($n = 50$): improved totals of features N^*.

			optimal cascade			
no.	N	N^*	(exhaustive search, using N^*	$E(n)$	no. of cascades	search time [s]
1	29	27	$(2,3,4,6,12)$	6.97378	14950	5.125
2	42	41	$(2,3,5,9,22)$	7.58432	91390	39.281
3	30	27	$(2,2,3,4,6,10)$	6.59502	65780	23.093
4	43	41	$(2,3,3,5,9,19)$	7.0788	658008	289.906

Table 5: Resulting cascades found numerically via continuous approximate expectations (4) and (5).

				resulting cascade		resulting cascade	
			optimal cascade	for approximate criterion (4)		for approximate criterion (5)	
n	K	A	(exhaustive search)	(`NMinimize[·]`)	time [s]	(`NMinimize[·]`)	time [s]
30	2	10^{-2}	$(7,16)$	$(7,16)$	0.609	$(7,16)$	0.620
30	2	10^{-3}	$(8,20)$	$(8,20)$	0.625	$(8,20)$	0.625
30	3	10^{-2}	$(4,6,14)$	$(4,6,14)$	2.688	$(4,6,14)$	2.797
30	3	10^{-3}	$(4,7,17)$	$(4,7,17)$	2.750	$(4,7,17)$	2.800
30	4	10^{-2}	$(3,4,6,10)$	$(3,4,6,10)$	6.484	$(3,4,6,10)$	6.422
30	4	10^{-3}	$(3,4,7,15)$	$(3,4,7,15)$	6.781	$(3,4,7,15)$	6.703
50	5	10^{-2}	$(2,3,4,7,13)$	$(2,3,4,7,13)$	18.953	$(2,3,4,7,13)$	20.001
50	5	10^{-3}	$(2,3,5,9,23)$	$(2,3,5,9,23)$	19.313	$(2,3,5,9,23)$	19.578
50	6	10^{-2}	$(2,2,3,4,7,12)$	$(2,2,3,4,7,12)$	26.141	$(2,2,3,4,7,12)$	26.875
50	6	10^{-3}	$(2,3,3,5,9,21)$	$(2,3,4,5,9,20)$	26.922	$(2,3,4,5,9,20)$	27.297

disposal gets combinatorially distributed among the $(n_1, \ldots, n_K)$ counts, (ii) the dependency of false alarm rates a_k on n_k counts is in general unknown, and obviously not continuous.

In this section we present a technique, tailored to the “cuboid in the corner” example, that allows to find optimal cascades via direct numerical optimization. This is achieved by a trick introducing a *continuous* approximate variant of formula (8) for $\mathrm{FAR}(m;m_0)$. In consequence, this further allows for continuous variants of formulas (4) and (5) for $E(n)$.

First, let us write down an equivalent representation of (8) as follows.

$$\mathrm{FAR}(m;m_0) = \prod_{k=1}^{n} w_k^{[1+m_0\leq k]\cdot[k\leq m_0+m]} \left(1-\prod_{k=1}^{n} w_k^{[1+m_0+m\leq k]\cdot[k\leq n]}\right) \Bigg/ \left(1-\prod_{k=1}^{n} w_k^{[1+m_0\leq k]\cdot[k\leq n]}\right). \qquad (11)$$

Note that all the products iterate now from $k=1$ to $k=n$ but the actual restrictions on k indexes are moved to exponents in the form of suitable indicator functions.

Consider the sigmoid function $\phi_\beta(x) = \frac{1}{1+e^{-\beta x}}$ with β parameterizing its steepness. Now, the key trick we apply is that indicator functions can be approximated by the sigmoid function for sufficiently large β in the following manner:

$$[a \leq b] \approx \phi_\beta(b-a+\frac{1}{2}). \qquad (12)$$

The $\frac{1}{2}$ is added because the inequality $a \leq b$ is not strict. Taking advantage of the trick we write down the following approximate version of the FAR formula:

$$\mathrm{FAR}(m;m_0) \approx \prod_{k=1}^{n} w_k^{\phi_\beta(k-1+m_0+\frac{1}{2})\phi_\beta(m_0+m-k+\frac{1}{2})} \left(1-\prod_{k=1}^{n} w_k^{[1+m_0+m\leq k]\cdot[k\leq n]}\right) \Bigg/ \left(1-\prod_{k=1}^{n} w_k^{[1+m_0\leq k]\cdot[k\leq n]}\right). \qquad (13)$$

The right-hand-side of (13) can be now plugged into the expectation formulas (4) or (5), and the numerical optimization can be carried out directly.

Table 5 reports cascades found numerically using Wolfram Mathematica's `NMinimize[·]` procedure performing an optimization with constraints. The constraints in our case were: $n_1+\cdots+n_K = N$ and $n_k \in \mathbb{N}$. The steepness parameter β was set to 10.0. The table shows that except for a single case, the numerical procedure found exactly the same cascades as the exhaustive combinatorial search.

4 EXAMPLE 2: "3D CUBE TRAP"

In this section we present the second geometric example, demonstrating the non-optimality of the standard VJ approach. The example is exactly three-dimensional. The domain is constituted by a cube that contains positive, negative and mixed regions inside of it, as depicted in Fig. 2. By mixed regions we mean the regions that cannot be separated by splits orthogonal to axes in the given space.

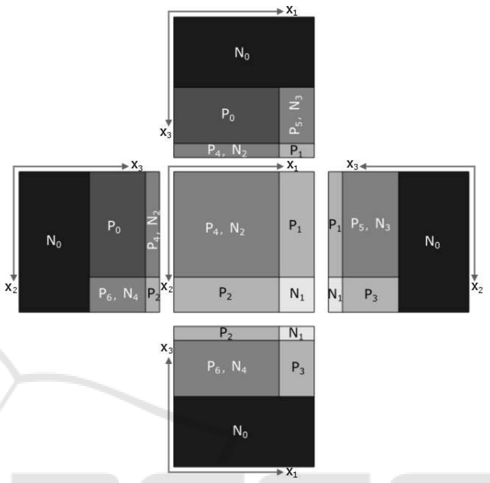

Figure 2: Example 2 "3D cube trap": visualization.

In this example the probability distribution in the cube is not uniform. From now on, we shall write $\mu(P_i)$ or $\mu(N_i)$ to denote the probability measure lying in the given region. In the example, the following particular values are imposed (but variations are possible without damaging the conclusions):

$$\begin{aligned}
p = \mu(P) &= 0.01,\\
1-p = \mu(N) &= 0.99,\\
\mu(P_1) &= 0.98\,\mu(P),\\
\mu(P_1) = \mu(P_2) = \mu(P_3) &= 0.002\,\mu(P),\\
\mu(P_4) &= 0.006\,\mu(P),\\
\mu(P_5) = \mu(P_6) &= 0.004\,\mu(P),\\
\mu(N_0) &= 0.975\,\mu(N),\\
\mu(N_1) &= 0.002\,\mu(N),\\
\mu(N_2) &= 0.001\,\mu(N),\\
\mu(N_3) = \mu(N_4) &= 0.011\,\mu(N).
\end{aligned}$$

Similarly as in the first example, the decision stumps are used as weak classifiers within the cascade training, but as a simplification suppose the splits can be carried out only along the borders of P_i, N_i regions.

We impose the following sensitivity / FAR constraints for the entire cascade: $D = 0.9$,

$A = 0.02$, $K = 2$. This results in $d_{\min} \approx 0.9486833$, $a_{\max} \approx 0.1414214$ serving as the per-stage requirements.

Description of the Resulting VJ-style Cascade

Stage 1: The feature x_3 is selected because of the large probability measure within the N_0 region. Regions lying above N_0 become labeled by the classifier as positives, whereas N_0 itself becomes labeled as negative. Other choices will significantly increase FAR or decrease sensitivity forcing the use of another feature. Because all P_i regions are classified as positives then $d_1 = 1$ and $a_1 = 0.025$.

Stage 2: At this stage we can either: (i) cut the cube orthogonally to x_1, x_2, (ii) try using x_3 again, or (iii) use some combination of two features. After the rejection of N_0 region, the remaining measure of negatives will be equal 0.025 due to the cascade properties.

Suppose the x_3 feature is selected again. By that we do not increase the number of features used so far. The classifiers labels the following regions as positive: $P_0, P_3, P_5, P_6, N_3, N_4$, the remaining ones become labeled as negatives. The following results are obtained.

$$\begin{aligned} d_2 &= \frac{0.98 + 2\cdot 0.004 + 0.002}{1} = 0.99 > d_{\min}, \\ a_2 &= \frac{2\cdot 0.011}{0.025} = 0.88 > a_{\max}, \\ a_1 \cdot a_2 &> A. \end{aligned}$$

Both the final and the per-stage requirements are not satisfied.

Suppose the x_1 feature is selected. The following regions will be marked as positives: $P_0, P_2, P_4, P_6, N_2, N_4$, other as negatives.

$$\begin{aligned} d_2 &= 0.9992 > d_{\min}, \\ a_2 &= 0.48 > a_{\max}, \\ a_1 \cdot a_2 &< A. \end{aligned}$$

Suppose the x_2 feature is selected. In this case, the results are be identical to the ones for x_1, because of the symmetry in the cube.

In both last cases the final requirements for FAR and sensitivity are satisfied and the learning process could be completed. However, the per-stage requirement for FAR is not met ($a_2 > a_{\max}$). Therefore, the algorithm following strictly the per-stage requirements will have to add one more feature.

Suppose the pair of features x_1, x_2 is selected. If we want to minimize FAR, the following regions will be marked as positives: P_0, P_4, N_2, other as negatives. Even if some different weights are assigned to each weak classifier, this will increase sensitivity at the expense of FAR. The results for this case are:

$$\begin{aligned} d_2 &= 0.986 > d_{\min}, \\ a_2 &= 0.04 < a_{\max}, \\ d_1 \cdot d_2 &> D, \\ a_1 \cdot a_2 &< A. \end{aligned}$$

At this point the learning process will be completed, since both final and per-stage requirements are satisfied.

Suppose the pair of features x_1, x_3 is selected (the option x_2, x_3 is symmetrical). If we want to minimize FAR, the following regions will be marked as positives: P_0, P_6, N_4, other as negatives.

$$\begin{aligned} d_2 &= 0.984 > d_{\min}, \\ a_2 &= 0.44 > a_{\max}. \end{aligned}$$

This combination does not meet the per-stage requirements.

To summarize, we remark that fulfilling the per-stage requirements implies the need an additional feature. As a result, the final cascade with $(n_1, n_2) = (1, 2)$ was achieved, using x_3 (stage 1) and x_1, x_2 (stage 2). Instead, the genuinely optimal cascade should be $(n_1, n_2) = (1, 1)$, using x_3 (stage 1) and either x_1 or x_2 (stage 2).

Let us now calculate the expected number of features for the cascade falling into the trap and compare it against an expectation for a cascade omitting it.

Expected Number of Features for the VJ-style Cascade

$$\begin{aligned} E(n) &= n_1 + n_2(pd_1 + (1-p)a_1) \\ &= 1 + 2(0.01\cdot 1.0 + 0.99\cdot 0.025) \\ &= 1.0695 \end{aligned}$$

Expected Number of Features for a Cascade Omitting the Trap

$$\begin{aligned} E(n) &= n_1 + n_2(pd_1 + (1-p)a_1) \\ &= 1 + 1(0.01\cdot 1.0 + 0.99\cdot 0.025) \\ &= 1.0348 \end{aligned}$$

5 EXAMPLE 3: "CHESSBOARD TRAP"

In this section we consider the last geometric example. The input domain is again constituted by a cube, this time n-dimensional, but the majority of decision boundaries is visible in the the x_1, x_2 subspace, as illustrated in Fig. 3.

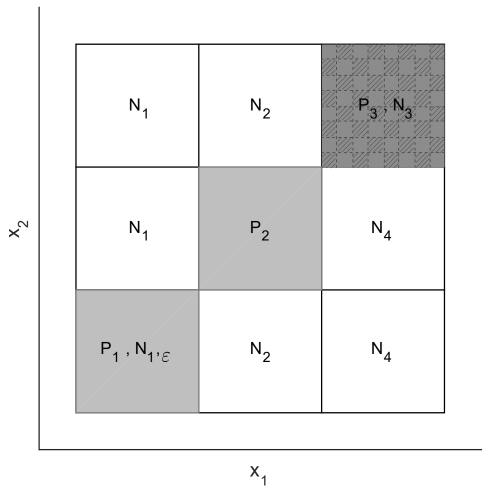

Figure 3: Example 3 "chessboard trap": visualization of x_1, x_2 subspace.

The set of positives is defined as follows:

$$
\begin{aligned}
P =& P_1 \cup P_2 \cup P_3, \\
P_1 =& \Big\{ (x_1, \ldots, x_n) \in [0,1]^n \colon 0 \leq x_1, x_2 < \frac{1}{3} \\
& \quad \text{and } 0 \leq x_3 < 1 - \varepsilon \Big\}, \\
P_2 =& \Big\{ (x_1, \ldots, x_n) \in [0,1]^n \colon \frac{1}{3} \leq x_1, x_2 < \frac{2}{3} \Big\}, \\
P_3 =& \Big\{ (x_1, \ldots, x_n) \in [0,1]^n \colon \frac{2}{3} \leq x_1, x_2 \leq 1 \\
& \quad \text{and } \mathrm{chess}(x_3, \ldots, x_n; 8) = 1 \Big\},
\end{aligned}
$$

where:

$$\mathrm{chess}(z_1, \ldots, z_q; m) = (\lfloor z_1 \cdot m \rfloor + \ldots + \lfloor z_q \cdot m \rfloor) \bmod 2.$$

The negative set is $[0,1]^n \setminus P$, but for clarity we define it explicitly as follows:

$$
\begin{aligned}
N =& N_1 \cup N_{1,\varepsilon} \cup N_2 \cup N_3 \cup N_4, \\
N_1 =& \Big\{ (x_1, \ldots, x_n) \in [0,1]^n \colon 0 \leq x_1 < \frac{1}{3} \\
& \quad \text{and } \frac{1}{3} \leq x_2 \leq 1 \Big\}, \\
N_{1,\varepsilon} =& \Big\{ (x_1, \ldots, x_n) \in [0,1]^n \colon \leq x_1, x_2 < \frac{1}{3} \\
& \quad \text{and } 1 - \varepsilon \leq x_3 \leq 1 \Big\}, \\
N_2 =& \Big\{ (x_1, \ldots, x_n) \in [0,1]^n \colon \frac{1}{3} \leq x_1 < \frac{2}{3} \\
& \quad \text{and } (0 \leq x_2 < \frac{1}{3} \text{ or } \frac{2}{3} \leq x_2 \leq 1) \Big\} \\
N_3 =& \Big\{ (x_1, \ldots, x_n) \in [0,1]^n \colon \frac{2}{3} \leq x_1, x_2 \leq 1 \\
& \quad \text{and } \mathrm{chess}(x_3, \ldots, x_n; 8) = 0 \Big\} \\
N_4 =& \Big\{ (x_1, \ldots, x_n) \in [0,1]^n \colon \frac{2}{3} \leq x_1 \leq 1 \\
& \quad \text{and } 0 \leq x_2 < \frac{2}{3} \Big\}
\end{aligned}
$$

Once again suppose that splits orthogonal to axes shall be carried out. As the reader may recognize, the arrangement of the example allows to: isolate P_2 using just two features (x_1 and x_2), isolate P_1 using three features (x_1, x_2 and x_3), but in order to perfectly isolate P_3 all n features must per force be used. This is caused by the fact that positives within P_3 together with negatives within N_3 are entangled in a chessboard pattern defined on all remaining variables $x_3, \ldots, x_n$. This sets the trap that the greedy stage-wise cascade training can fall into.

The probability measures are distributed as described below (variations are possible) with two parameters ε and α that we shall discuss later on:

$$
\begin{aligned}
p = \mu(P) &= 0.01, \\
1 - p = \mu(N) &= 0.99, \\
\mu(P_1) &= 0.05\, \mu(P), \\
\mu(P_2) &= 0.85\, \mu(P), \\
\mu(P_3) &= 0.10\, \mu(P), \\
\mu(N_1 \cup N_{1,\varepsilon}) &= 0.9\, \mu(N), \\
\mu(N_{1,\varepsilon}) &= \varepsilon\, 0.9\, \mu(N), \\
\mu(N_2 \cup N_3 \cup N_4) &= 0.1\, \mu(N), \\
\text{with } \mu(N_3) &= \alpha\, 0.1\, \mu(N) \quad (\text{e.g. } \alpha \geq 0.9).
\end{aligned}
$$

Please note that we impose a large α fraction, so that the chessboard region encapsulates the large part of the $0.1\mu(N)$ negatives probability measure remaining outside of N_1 and $N_{1,\varepsilon}$.

Algorithm 3 presents the pseudocode of a cascade training algorithm suited to the mathematical properties of the considered example. Again, the training is guided by the stage-wise sensitivity / FAR requirements (VJ-style). At each stage, the algorithm successively tries out more complex classifiers based on orthogonal splits. It iterates first over single features, then pairs of features, then triples, and so forth. For a fixed combination of features, the algorithm constructs a decision tree which can apply *any* number of splits (but using only the given features) in order to minimize the classification error. In particular, the cuboids isolated by the splits can be labeled arbitrarily as fits. When a certain combination of features allows to satisfy stage requirements $d_{\min}$ and $a_{\max}$, the algorithm ceases searching further and the current cascade stage becomes closed.

Algorithm 3: Viola Jones-style cascade training based on stage-wise sensitivity / FAR requirements (for the purposes of Example 3).

```
procedure TRAINCASCADEVJSTYLE($D$, $A$, $K$)
    $d_{\min} := D^{1/K}$, $a_{\max} := A^{1/K}$,
    for $k := 1,\dots,K$ do
        $n_k := 0$, $d_k := 0$, $a_k := 1$
        for $d := 1,\dots,n$ do
            for all feature combinations $x_{i_1},\dots,x_{i_d}$
            of length $d$ $(1 \le i_1 < \cdots < i_d \le n)$ do
                build the decision tree using
                    features $x_{i_1},\dots,x_{i_d}$ that
                    minimizes classification error
                    (any number of splits allowed)
                calculate $d_k, a_k$ for the tree
                memorize the tree if it is better than
                    the best tree so far (for curred $d$)
            end for
            if ($d_k \ge d_{\min}$ and $a_k \le a_{\max}$) then
                use the best tree for current stage
                $n_k := d$
                jump out of this loop
            end if
        end for
    end for
end procedure
```

Suppose the following requirements for the entire cascade have been imposed: $K = 2$, $D = 0.9025$, $A = 0.01$. This implies: $d_{\min} = D^{1/K} = 0.95$, $a_{\max} = A^{1/K} = 0.1$.

Description of the Resulting VJ-style Cascade
Stage 1: It is possible to see that the single feature x_1 will turn out to be sufficient, producing the following classifier for the first stage:

$$F_1(\mathbf{x}) = \begin{cases} -1, & \text{for } 0 \le x_1 < \frac{1}{3}; \\ 1, & \text{otherwise.} \end{cases}$$

We obtain $d_1 = 0.95$ and $a_1 = 0.1$ for this classifier, which satisfies stage requirements. Therefore, $n_1 = 1$.
Stage 2: We will now prove that, perforce, all the remaining features shall be required for the second stage. The reasoning is as follows. To satisfy stage requirements the next classifier should perfectly isolate the P_2 region (computationally cheap because only one new feature x_2 is added) but also some fragment, say $\beta \in (0,1)$, from the P_3 region. In order to achieve $d_2 \ge d_{\min}$, the following condition must be met

$$\frac{\mu(P_2) + \beta\,\mu(P_3)}{\mu(P_2) + \mu(P_3)} \ge d_{\min},$$

which yields $\beta \geqslant 0.525$. Note that if one does not apply all $n-2$ remaining features $(x_3,\dots,x_n)$, but only some subset of them, then the chessboard pattern can never be perfectly recognized (that is why the chessboard is convenient for this example). In turn, when some non-perfect split is made, such that the $\beta \cdot \mu(P_3)$ measure is kept at one side then it simultaneously introduces $\beta \cdot \alpha \cdot \mu(N_3)$ of false alarms. In other words, the following condition:

$$\frac{\beta\,\alpha\,0.1\mu(N)}{0.1\mu(N)} \le a_{\max},$$

yields $\alpha \le 0.1/0.525$ which is not allowed according to former settings ($\alpha \ge 0.9$). Therefore, all the remaining features $x_3,\dots,x_n$ and x_2 are required and the classifier for the second stage is

$$F_2(\mathbf{x}) = \begin{cases} 1, & \text{for } \frac{1}{3} \le x_1, x_2 < \frac{2}{3}; \\ 1, & \text{for } \frac{2}{3} \le x_1, x_2 \le 1 \\ & \text{and } \operatorname{chess}(x_3,\dots,x_n; 8) = 1; \\ -1, & \text{otherwise;} \end{cases}$$

yielding $d_2 = 1, a_2 = 0.0$

Let us now calculate explicitly the expected number of features for the cascade falling into the chessboard trap and compare it against an expectation for a cascade omitting it.

Expected Number of Features for the VJ-style Cascade

$$\begin{aligned} E(n) &= n_1 + n_2(pd_1 + (1-p)a_1) \\ &= 1 + (n-1)(0.01 \cdot 0.95 + 0.99 \cdot 0.1) \\ &= 1 + 0.1085(n-1) \end{aligned}$$

Expected Number of Features for a Cascade Omitting the Trap
To omit the trap it is sufficient to classify the whole chessboard region $P_3 \cup N_3$ as negative at stage one. This could be achieved for example by the following cascade.

$$F_1(\mathbf{x}) = \begin{cases} 1, & \text{for } 0 \le x_1, x_2 < \frac{1}{3}; \\ 1, & \text{for } \frac{2}{3} \le x_1, x_2 \le 1; \\ -1, & \text{otherwise.} \end{cases}$$

$$F_2(\mathbf{x}) = \begin{cases} -1, & \text{for } 0 \leq x_1, x_2 < \frac{1}{3} \\ & \text{and } x_3 \geq 1 - \varepsilon; \\ 1, & \text{otherwise.} \end{cases}$$

Hence, for stage 1 we have: $n_1 = 2$, $d_1 = 0.9$, $a_1 = \varepsilon\, 0.9$. Whereas, for stage 2 we have: $n_2 = 1$ because x_3 is the only new variable, $d_2 = 1$, $a_2 = 0$. We remark that cutting of the ε-region is required if

$$\frac{\varepsilon\, 0.9\mu(N)}{\mu(N)} > A,$$

which yields $\varepsilon > 0.01$.

The expected number of features for the above cascade is

$$\begin{aligned} E(n) &= n_1 + n_2(pd_1 + (1-p)a_1) \\ &= 2 + 1(0.01 \cdot 0.9 + 0.99\varepsilon \cdot 0.9) \\ &= 2.009 + 0.891\varepsilon \\ &=_{|\varepsilon:=0.02}\ 2.02682. \end{aligned}$$

Comparing the two cascades we see that the second one becomes better if

$$1 + 0.1085(n-1) > 2.02682,$$

which happens whenever the number of features (dimensions) is $n \geq 11$.

6 CONCLUSIONS

We have provided three geometric examples demonstrating how training a cascade with sensitivity / FAR constraints imposed per each stage can lead to non-optimality in the cascade computational cost — more precisely, in the expected number of features $E(n)$ the cascade uses while performing a detection procedure.

We are aware that the constructed examples are artificial and motivated by certain mathematical purposes. Yet, similar properties or variations can be easily met in real-world data. By generalizing the disadvantages of the stage-wise training pointed out by our examples, the following two conclusions can be formulated.

1. A stage satisfying the $(d_{\min}, a_{\max})$ requirements is in some cases closed prematurely, which may result in a greater number of new features required by the stages to follow. Investing more features in such a stage (though not greedy) can in some cases be beneficial, lowering the final $E(n)$ value.
2. There exist many cases where a stage can be closed without actually satisfying the $(d_{\min}, a_{\max})$ requirements, and still the final requirements for the entire cascade can be met.

The conclusions above can help to avoid the pitfalls caused by the greedy approach guided by the stage-wise constraints.

In our further research we plan to design specific cascade training algorithms based partially on search methods and keeping track of more than one cascade in runtime. Let us briefly sketch this idea hereby. Training a cascade can be brought to a graph searching process with nodes representing cascade stages with slightly different counts of features. For example, in the simplest case one could consider three nodes per each stage: a node using exactly the number of features suggested by the traditional approach, a node with one feature more (this worsens the expected value of features up to the current stage but improves the sensitivity / FAR properties), and a node with one feature less (this improves the expected value but worsens the sensitivity / FAR properties). By tracing the ‘evolution’ of such additional nodes further on one may hope to discover cascades with improved overall expected values of features.

ACKNOWLEDGEMENTS

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

REFERENCES

Bourdev, L. and Brandt, J. (2005). 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 - Volume 02*, CVPR '05, pages 236–243. IEEE Computer Society.

Crow, F. C. (1984). Summed-area Tables for Texture Mapping. *SIGGRAPH Comput. Graph.*, 18(3):207–212.

Freund, Y. and Schapire, R. (1996). Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156. Morgan Kaufman.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 28(2):337–407.

Gama, J. and Brazdil, P. (2000). Cascade Generalization. *Machine Learning*, 41(3):315–343.

Li, J. and Zhang, Y. (2013). Learning SURF Cascade for Fast and Accurate Object Detection. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '13, pages 3468–3475. IEEE Computer Society.

Papageorgiou, C. P., Oren, M., and Poggio, T. (1998). A general framework for object detection. In *Computer*

Vision, 1998. Sixth International Conference on, pages 555–562.

Pham, M. and Cham, T. (2007). 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*, pages 1–7.

Saberian, M. and Vasconcelos, N. (2014). Boosting Algorithms for Detector Cascade Learning. *Journal of Machine Learning Research*, 15:2569–2605.

Shen, C., Wang, P., Paisitkriangkrai, S., and van den Hengel, A. (2013). Training Effective Node Classifiers for Cascade Classification. *International Journal of Computer Vision*, 103(3):326–347.

Shen, C., Wang, P., and van den Hengel, A. (2010). Optimally Training a Cascade Classifier. *CoRR*, abs/1008.3742.

Vallez, N., Deniz, O., and Bueno, G. (2015). Sample selection for training cascade detectors. *PLos ONE*, 10.

Viola, P. and Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. In *Conference on Computer Vision and Pattern Recognition (CVPR'2001)*, pages 511–518. IEEE.

Viola, P. and Jones, M. (2004). Robust Real-time Face Detection. *International Journal of Computer Vision*, 57(2):137–154.