11

Real-Time Algorithms of Object Detection Using Classifiers

Roman Juránek, Pavel Zemčík and Michal Hradiš
Graph@FIT
Faculty of Information Technology, Brno University of Technology
Czech Republic

1. Introduction

Object detection, or more generally pattern detection and recognition, can be based on many different principles. The objects can be described through their structure, shape, color, texture, etc. [Blaschko & Lampert (2009); Chen et al. (2004); Fidler & Leonardis (2007); Leibe et al. (2008); Lowe (1999); Serre et al. (2005); Viola & Jones (2001)]; therefore, a variety of object detection mechanisms was developed over time. One of the modern approaches to object detection is similarity-based detection where the objects of interest are defined through a set of examples and typically also through a set of counter-examples and the decision whether an object is an object of interest is done through machine learning-based functional block – classifier. The object detection in an image is performed by the application of the classifier on sub-windows of the image.

The focus in this chapter is on statistical binary classifiers whose function is to make a binary decision on whether an image region is or is not an object of interest. The methods of interest include mainly AdaBoost [Freund (1995); Schapire et al. (1998)] whose original purpose was to fuse a small number of relatively well working so-called *weak hypotheses* into one, better working, *strong classifier*. This approach was further developed into an approach, which instead of a small number of weak classifiers, took into account a large number of simple functions and selected suitable weak classifiers automatically from these functions. This method has been demonstrated in the pioneer work of Viola and Jones [Viola & Jones (2001)].

The AdaBoost approach has been further refined and modified [Bourdev & Brandt (2005); Li et al. (2002); Sochman & Matas (2004; 2005)]. Perhaps the most important modification was by Sochman & Matas (2005), called WaldBoost which was based on Wald's sequential decision making [Wald (1947)] combined with AdaBoost. The main advantage of WaldBoost is its significant performance gain comparing it to the AdaBoost classifiers with virtually no change in classification quality.

The detection through classification involves the application of the classifier on a selection of sub-images of the analyzed image. As the classification results of neighboring sub-images may be statistically significantly interdependent, it is worth studying whether the inter-dependencies can be exploited to reduce the computational effort through the prediction of classifier results in certain sub-images, through suppression of unwanted object detection

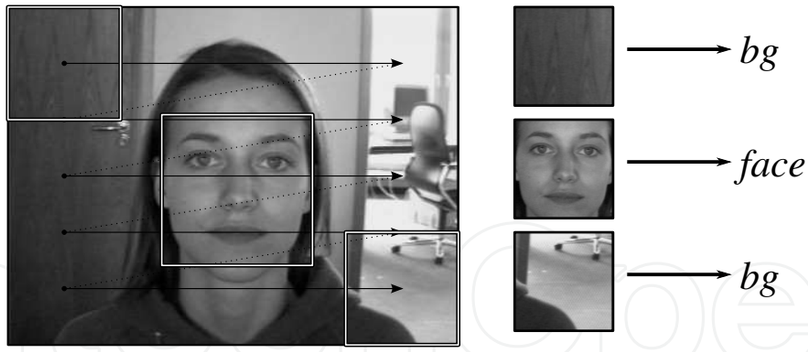

Fig. 1. Scanning the image with a classifier. Individual sub-images of the image are classified by a classifier (Image source: BioID dataset).

(e.g. multiple detections in very close image locations), or simply through the sharing of intermediate results of the calculations. These aspects of object detection are addressed in this chapter as well.

The structure of the chapter is as follows. The next section gives a brief introduction to object detection with classifiers. Section 3 discusses properties of features extracted from image and describes feature types often used for rapid object detection. Section 4 describes the ideas behind AdaBoost and WaldBoost learning procedures. Acceleration methods for WaldBoost-based detection are introduced in Section 5. Implementation of the detection runtime on different platforms is discussed in Section 6. Some results of the detection acceleration are presented in Section 7, and finally we conclude in Section 8 with some ideas for future research.

2. Object detection with classifiers

Classifiers are suitable for making the decision, whether some sub-images are images of object of interest or not. Such functionality is obviously of interest for object detection but it is not sufficient on its own. The reason is that for reliable classification, variability of objects of interest has to be minimized - the classifiers are trained to detect well-aligned, centered and size-normalized objects in the classified sub-image. Therefore, the actual detection of objects is performed through a classification of contents of all the sub-windows that can contain the object of interest, or simply through classification of all the possible sub-windows. This is usually performed by *scanning* the image with a moving window of a fixed size where the content of the window is classified for each location and, if the object of interest is found, the location is considered the output of the detection process.

The above described approach involves, in fact, an exhaustive search for an object of interest in the image, where all the sub-images are classified in order to understand whether they contain an object of interest or not. While the classification process is in general quite simple (as shown in more detail below), sometimes it might be feasible to pre-process the analyzed image in order to identify the image parts where the object(s) of interest cannot be present; such parts of the image can be excluded from the classification process and the computational effort can be reduced. Good examples of such approach are color-based pre-processing, where e.g. a flower cannot be present in a part of the image that contains “completely blue sky”; or a human face cannot be found in a part of an image that does not contain “skin color”; or

geometry-based approaches where it cannot be expected that an airplane would be detected below walking people in the image.

As it is obvious from the above description, detection of objects through AdaBoost/WaldBoost methods is dependent on object orientation and size; however, in many applications it is desirable to detect objects regardless of their size or orientation. While this requirement is difficult or often impossible to handle directly in the AdaBoost/WaldBoost machine learning process, the feasible approach is to handle it indirectly through repeating the detection process for different scales and/or orientations. The main reason is that in general, the feature extraction methods (weak classifiers) are not rotation, scale or shift invariant. Therefore, the detection process should be applied repeatedly to *sample* the rotation, scale, etc. in the needed range. The density of image sampling is dependent on the tolerance of the classifier to rotation, scale, etc. The tolerance is in general not predictable and depends on the dataset.

3. Efficient feature extraction

The performance of the object detection is for the large part influenced by underlying feature extraction methods. Two main properties of features extracted from an image exist: *a)* descriptive power and *b)* computational complexity. The goal in rapid object detection is to use computationally simple and, at the same time, descriptive features. In the vast majority of cases, these two properties are mutually exclusive and thus there are computationally simple features with low descriptive power (e.g. isolated pixels, sums of area intensity) or complex and hard to compute features with high descriptive power (Gabor wavelets [Lee (1996)], HoG [Dalal & Triggs (2005)], SIFT and SURF [Bay et al. (2008); Lowe (2004)], etc.). A close to ideal approach is Viola and Jones [Viola & Jones (2001)] with their Haar features calculated in constant time from an integral representation of image. The features used in this chapter are Local Binary Patterns (LBP) [Zhang et al. (2007)], Local Rank Patterns (LRP) [Hradiš et al. (2008)] and Local Rank Differences (LRD) [Zemcik et al. (2007)]. Their main properties are as follows.

- *Strict locality* – Evaluation is based strictly on local data (i.e. no normalization is needed).
- *Simple evaluation* – The input is coefficients extracted from an image by convolution with a rectangular kernel. The coefficients are processed by a simple formula.

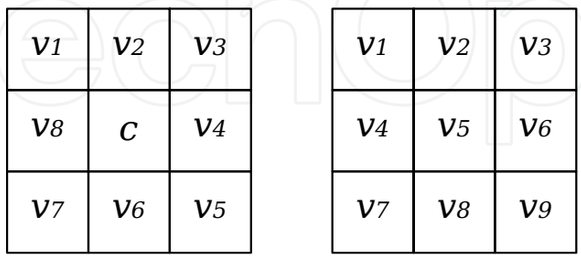

Fig. 2. Feature samples for LBP (left), LRD and LRP (right)

All presented features are based on the same model. The only difference is their evaluation function. First, coefficients v_i from regular 3×3 grid (see Fig. 2) are extracted by convolution. The coefficients are processed by an evaluation function producing the response.

$$LBP(\mathbf{v}, c) = \sum_{i=0}^{N} (v_i > c) 2^i \tag{1}$$

$$LRD(\mathbf{v}, a, b) = r(v_a, \mathbf{v}) - r(v_b, \mathbf{v}) \tag{2}$$

$$LRP(\mathbf{v}, a, b) = 10r(v_a, \mathbf{v}) + r(v_b, \mathbf{v}) \tag{3}$$

The evaluation of LBP works such that all samples are compared to the central one. The result of each comparison is treated as a single bit in the 8 bit code (1). The LRD and LRP features are parametrized by indices of two samples whose ranks are calculated (4). The ranks are subtracted in the case of LRD or combined together in LRP (2,3).

$$r(v, \mathbf{v}) = \sum_{i=1}^{9} \begin{cases} 1, \text{when } v > v_i \\ 0, \text{otherwise} \end{cases} \tag{4}$$

The response range of the features is $\langle 0, 255 \rangle$ for LBP, $\langle -8, 8 \rangle$ for LRD and $\langle 0, 99 \rangle$ for LRP. The response is used as an input to a weak classifier which is essentially a look-up table assigning a weak classifier response to a feature response.

4. AdaBoost and WaldBoost

AdaBoost [Freund (1995)] and other boosting algorithms [Friedman et al. (2000); Grove & Schuurmans (1998); Ratsch (2001); Rudin et al. (2004); Schapire et al. (1998)] all combine *weak hypotheses* $h_t : \chi \to \mathbb{R}$ into a *strong classifier* H_t. The combination is a weighted average where responses of the weak hypotheses are multiplied by weights α determining their importance:

$$H_T(\mathbf{x}) = \sum_{t=1}^{T} (h_t(\mathbf{x})) \tag{5}$$

The weak hypotheses often internally partition the object space $\boldsymbol{\emptyset}$ into a set of disjoint areas based on a single feature response. Such weak hypotheses are called space partitioning weak hypotheses [Schapire & Singer (1999)] and the partition functions $f : \chi \to \mathbb{N}$ are referred to in the following text simply as *features*. The weak space partitioning hypotheses are combinations of such features and a *look-up table function* $l : \mathbb{N} \to \mathbb{R}$

$$h_t(\mathbf{x}) = l_t(f_t(\mathbf{x})). \tag{6}$$

The real value assigned by l_t to output j of f_t is denoted as $c_t^{(j)}$ in the text.

Most of the boosting algorithms order the weak classifiers starting with the most informative one and thus it is reasonable to evaluate them in this order and stop when the classification decision is certain enough. Such classifiers are called *soft cascades* [Bourdev & Brandt (2005)] and can be formalized as a *sequential decision strategy* [Sochman & Matas (2005)] S which is a sequence of decision functions $S = S_1, S_2, \ldots, S_T$, where $S_t : \mathbb{R} \to \sharp, -1$. The evaluation of the strategy is terminated with a negative result when a decision function outputs -1. The decision functions S_t decide based on a tentative sum of the weak hypotheses $H_t, t < T$ which

is compared to a threshold θ_t:

$$S_t(\mathbf{x}) = \begin{cases} \sharp, & \text{if } H_t(\mathbf{x}) > \theta_t \\ -1, & \text{if } H_t(\mathbf{x}) \leq \theta_t \end{cases}. \tag{7}$$

WaldBoost [Sochman & Matas (2005)] is a method which produces an optimal decision strategy for a target false negative rate. The algorithm combines real AdaBoost Schapire & Singer (1999) and Wald's *sequential probability ratio test* Wald (1947).

Given a weak learner algorithm, training data $\{(x_1, y_1) \ldots, (x_m, y_m)\}, x \in \chi, y \in \{-1, +1\}$ and a target false negative rate α, the WaldBoost algorithm finds a decision strategy S^* with a miss rate α_S which is lower than α and the average evaluation time $\bar{T}_S = E(\arg\min_i(S_i \neq \sharp))$ is minimal:

$$S^* = \arg\min_S \bar{T}_S, \text{ s.t. } \alpha_S < \alpha.$$

WaldBoost uses real AdaBoost to iteratively select the most informative weak hypotheses h_t. The threshold θ_t is then selected in each iteration so that as many negative training samples are rejected as possible while asserting that the likelihood ratio that is estimated on training data

$$\hat{R}_t = \frac{p(H_t(\mathbf{x}) < \theta_t | y = -1)}{p(H_t(\mathbf{x}) < \theta_t | y = +1)}$$

satisfies $\hat{R}_t \geq \frac{1}{\alpha}$.

5. Acceleration of WaldBoost based object detection

Acceleration of object detection can be in general based on several principles, the key ones being:

- Implementation on a (more) powerful computational platform – simple general improvement of computational platforms
- exploitation of a structurally different platform compared to the traditional processor platform
- improvement of the AdaBoost/WaldBoost machine learning and/or feature extraction algorithms
- exploitation of redundancy and coherence in results of classification in different (adjacent or close) areas of the image.

The case of general improvement of computational platforms is not of interest here in this publication. On the other hand, structurally novel computational platforms are interesting in general due to their rapid growth in computer technology and specifically in the object detection, where the structure of exploitation of the computational resources suggests that the traditional platforms are not ideal and that the massive parallel platforms are also not completely suitable.

The general improvements of the AdaBoost/WaldBoost machine learning methods are outside the scope of this publication. However, the algorithmic improvements not connected with the classification itself, but rather with the redundancy due to correlation of the classification results in different sub-images of the same image, are quite important to

investigate. Their exploitation can significantly reduce the computational effort needed for object detection.

5.1 Classification cost and its minimization

The relative cost of classifier evaluation can be measured and used for the reduction of the computational effort by combining two or more different approaches of classifier implementation; for example, a hardware pre-processing unit connected to post-processing unit on traditional CPU. The minimization method can be applied to various types of relative cost (computations, memory, hardware price, etc.) as its formulation is general. In this chapter, the interest is in the minimization of the use of computational resources and the relative cost thus roughly corresponds to computational time (except when otherwise noted).

5.1.1 Classifier statistics

The main property of a classifier is the probability of the evaluation of a weak hypothesis, reflecting on how often a weak hypothesis is executed during the detection. This value p can be calculated for every stage i from statistics obtained on a dataset of images. Due to the rejection nature of WaldBoost classifiers, the sequence of p_i decreases and the first stage is always evaluated (i.e. the $p_1 = 1$). Example of such statistics is shown on the left in Fig. 3. The p_i captures computational the complexity of the classifier.

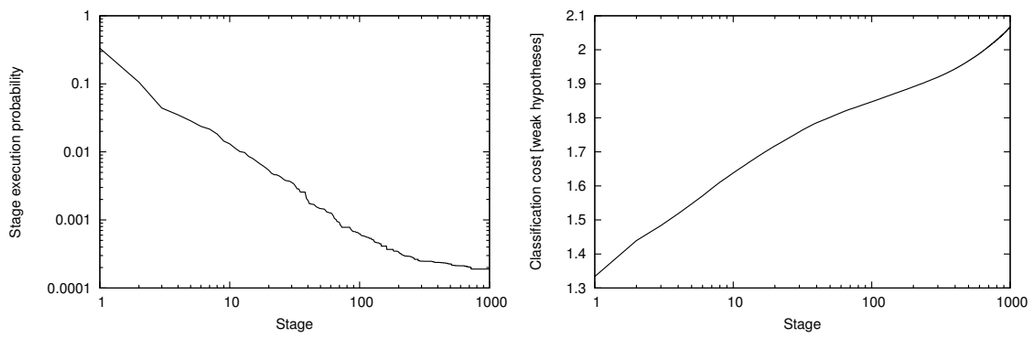

Fig. 3. Example of classifier statistics. Left, stage execution probability. Right, number of evaluated weak hypotheses on average for particular length of the classifier.

5.1.2 Cost evaluation

In the case of AdaBoost/WaldBoost classifiers the total cost C is proportional to the number of evaluated weak hypotheses which can be calculated by (8). The T is the length of classifier. The k is the overall classifier cost which symbolizes evaluation cost on a particular platform on which the classification is implemented. The p is the probability of the execution of a particular weak hypothesis (see Section 5.1.1). The c is the relative cost of the weak hypothesis evaluation which addresses the possibility that the hypotheses have a different cost (due to the use of different features, for example).

$$C = k \sum_{i=1}^{T} p_i c_i \tag{8}$$

When analyzing real classifiers, p can be obtained from the statistics on input images and c by time measurement or other cost estimation and k can be set to a constant value. In Fig 3, the left plot shows the value of p_i and the right plot the area under the p_i curve which is proportional to the amount of computational resources needed for the evaluation of the classifier.

In object detection, the most common are homogeneous classifiers (i.e. those with all weak classifiers and features of the same type). In such cases, the cost of hypothesis evaluation is constant $c_i = c$. Additionally in AdaBoost, all weak hypotheses are executed every time and the probability of executing all hypotheses is equal to $p_i = 1$. The C from (8) can thus be simplified to $C^{(AB)}$ (for AdaBoost) and $C^{(WB)}$ (for WaldBoost) in (9).

$$C^{(AB)} = knc \qquad C^{(WB)} = kc \sum_{i=1}^{n} p_i \tag{9}$$

5.1.3 Cost minimization

The cost of classification is not the only property of the classifier, but it is also the property of implementation of the run-time in which the classifier is executed – feature extraction and classifier evaluation. Different implementations with different properties exist. Imagine, for example, an implementation A which can evaluate very efficiently $K > 1$ weak hypotheses in a row, but it always evaluates *all* of them no matter how many weak hypotheses is actually needed for the evaluation. It could be a pre-processing unit implemented in a hardware which rejects areas without an occurrence of the target object. Then, there is implementation B in software which can evaluate the classifier in standard way. The computational cost for one feature in A is much less than in B but implementing the whole classifier in the hardware is hard to achieve due to limited resources.

$$C = \arg\min_{0 \le u \le T} \left(k_1 \sum_{i=0}^{u-1} p_{1,i} c_{1,i} + k_2 \sum_{i=u}^{T-1} p_{2,i} c_{2,i} \right) \tag{10}$$

Both implementations can be put together, but the problem is how many weak hypotheses have to be put in a hardware unit and how many are left in the software. The precise position of division of the evaluation is subject to minimization of classification cost (10) in order to find a composition with minimal cost.

The two-phase classifier can be fine tuned by one parameter. Equation 10 shows the minimization problem and Fig. 4 shows values of C for different settings of u. The C is the total minimal cost of the evaluation; u is the point of classifier division; and k, c and p correspond to the parameters of the cost computation from Equation 8. It should be noted, that although the properties p of the classifier are the same for both parts, the p can be in general different for each part. This is due to the structure of the evaluation in particular implementation which can *force* different probabilities of feature evaluation (e.g. by evaluating more features in one step; see Section 5.1.1).

When going beyond the example given above, more than two phases of evaluation can be used. And minimization problem is thus multi-dimensional. In the general case, described by (11), the classifier division is vector $\mathbf{u}$ whose values are searched for in order to find the best composition of parts with different properties. Note that u_i can be equal to u_{i+1} and some

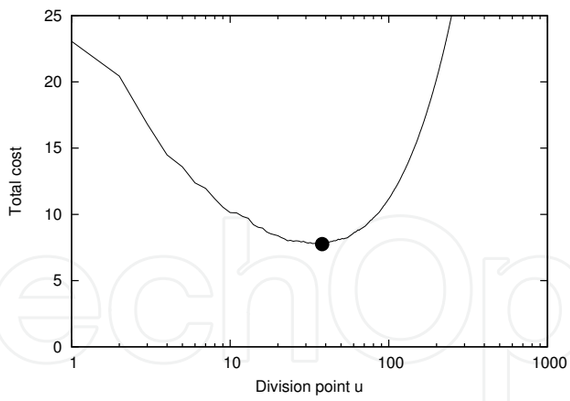

Fig. 4. Example of minimization of classification cost for two-phase classifier. The first phase always evaluates all weak hypotheses but the cost for a weak hypothesis is 0.1 of the second phase. The second phase evaluates weak hypotheses one by one. The black dot marks the division between the parts that lead into the minimum cost.

parts could be in fact skipped when they are evaluated as useless in the optimization.

$$\begin{aligned} & C = \arg\min_{\mathbf{u}} \left(\sum_{m=1}^{M} \left(k_m \sum_{i=u_{m-1}}^{u_m - 1} p_{m,i} c_{m,i} \right) \right) \\ & s.t. \\ & u_0 = 0 \\ & u_m = T \\ & u_{i-1} \leq u_i, \;\; 0 \leq i \leq M \end{aligned} \tag{11}$$

In practical applications, it is easy to get classifier statistics – it reflects classifier behavior on images. On the other hand, it is tricky to identify values of c and k. It has to be done by careful examination of performance of the particular implementation of the detection (e.g. by the precise measurement of time needed for the execution of weak hypotheses).

5.2 Exploiting neighbors

In scanning window object detection using a soft cascade detector, each image position is processed independently. However, much information is shared between neighboring positions and utilizing this information has a potential for increasing the speed of detection.

One way to utilize the shared information is to learn *suppression classifiers* [Zemčík et al. (2010)] to predict the responses of the original detection classifier at neighboring positions. Computation of the original detector can then be suppressed at positions for which this prediction is negative and with enough confidence.

In the case of *space partitioning weak hypotheses* (see Section 4), the suppression classifiers can be made computationally very efficient by re-using the features h_t computed by the original classifier. In that case, adding the suppression classifiers just increases the size of the look-up table $l : \mathbb{N} \rightarrow \mathbb{R}$.

The task of learning the suppression classifiers can be formulated as detector emulation [Šochman & Matas (2007); Sochman & Matas (2009)] which allows usage of

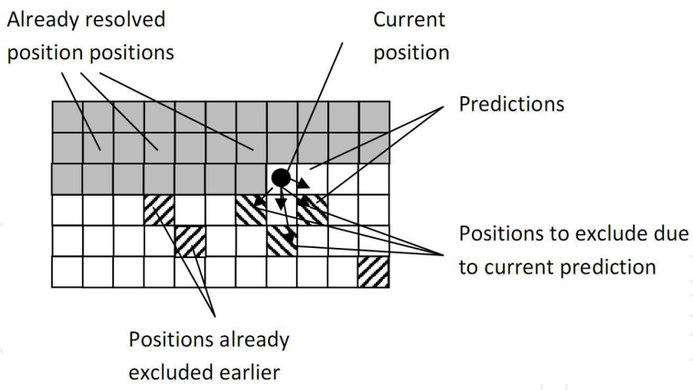

Fig. 5. Neighborhood suppression - during scanning, positions surrounding the currently evaluated position can be suppressed. On such positions the classifier will not be computed.

unlabeled data for training and does not require any modifications in learning the original detection classifier. Moreover, previously created detectors can be used as well.

In the classifier emulation [Šochman & Matas (2007); Sochman & Matas (2009)] approach, an existing detector is considered a black box and its decisions are used as labels for a new WaldBoost learning problem. The algorithm for learning the suppression classifiers differs from this basic scenario in three distinct aspects discussed below. The whole algorithm for learning suppression classifier is summarized in Algorithm 1.

The first change, as mentioned earlier, is that the weak hypotheses h'_t of a suppression classifier, reused features f_t of the original detector and only new look-up table functions l'_t are learned. By restricting the features, the learning process is very fast as the selection of an optimal weak hypothesis is generally the most time consuming step.

The second difference is that the labels for training the suppression classifier are obtained from a different image position than where the classifier gets information from (the position containing the original features l_t). This is consistent with the fact that we want to predict responses in the neighborhood of the currently evaluated position.

Finally, the set of training samples is pruned twice in each iteration of the learning algorithm instead of only once as in WaldBoost. The samples rejected by the new suppression classifier are removed from the training set, as well as, the samples rejected by the original classifier. This reflects the behavior during scanning when only those features which are needed by the detector to make a decision are computed and, consequently, the suppression classifiers can only use these computed features to make their own decision.

5.3 Early non-maxima suppression

Detection of objects by a scanning window technique usually employs some kind of *non-maxima suppression* to select a position with the highest classifier response from a small neighborhood in position, scale and other possible degrees of freedom. The suppressed detections have no influence on the resulting detection and it may not be necessary to compute the detectors completely in these positions. In other applications only the highest response on a number of samples is of interest as well. Examples of such applications are speaker

Algorithm 1 WaldBoost for learning suppression classifiers

Input: original soft cascade $H_T(x) = \sum_{t=1}^{T} h_t(x)$, its early termination thresholds $\theta'^{(t)}$ and its features f_t; desired miss rate α; training set $\{(x_1, y_1) \ldots, (x_m, y_m)\}, x \in \chi, y \in \{-1, +1\}$, where the labels y_i are obtained by evaluating the original detector H_T at an image position with a particular displacement with respect to the position of corresponding x_i

Output: look-up table functions l'_t and early termination thresholds $\theta'^{(t)}$ of the new suppression classifier

Initialize sample weight distribution $D_1(i) = \frac{1}{m}$

for $t = 1, \ldots, T$

1. estimate new l'_t such that its

$$c_t^{(j)} = -\frac{1}{2} \ln \left(\frac{Pr_{i \sim D}(f_t(x_i) = j | y_i = +1)}{Pr_{i \sim D}(f_t(x_i) = j | y_i = -1)} \right)$$

2. add l'_t to the suppression classifier

$$H'_t(x) = \sum_{r=1}^{t} l'_r(f_r(x))$$

3. find optimal threshold $\theta'^{(t)}$
4. remove from the training set samples for which $H_t(x) \leq \theta^{(t)}$
5. remove from the training set samples for which $H'_t(x) \leq \theta'^{(t)}$
6. update the sample weight distribution

$$D_{t+1}(i) \propto \exp(-y_i H'_t(x_i))$$

and person recognition where a short utterance or face image is matched by a classifier to templates from a database.

The main idea of *Early non-Maxima Suppression* [Herout et al. (2011)] (EnMS) is to perform non-maxima suppression already during computation of classifiers and to stop computing classifiers for objects having very low probability to reach the best score in the set of the competing objects.

In the context of soft cascades, EnMS can be formalized as the *Conditioned Sequential Probability Ratio Test* (CSPRT) which allows the decision functions S_t (see Equation 7 for the original formulation) to be conditioned by some additional data $z_t \in \mathcal{Z}$:

$$S_t(x, z_t) = \begin{cases} -1, \text{ if } H_t(x) < \theta_t(z_t) \\ \sharp, \quad \text{if } \theta_t(z_t) \leq H_t(x) \end{cases} \tag{12}$$

Here the threshold becomes a function of the conditioning data.

In order to create an optimal CSPRT strategy, the threshold functions $\theta_t(z_t)$ should be optimized for the same objectives as the thresholds θ_t in WaldBoost (see Equation 13). Parameters of $\theta_t(z_t)$ should be set so that as many negative training samples are rejected as

possible while asserting that the likelihood ratio is estimated on the training data

$$\hat{R}_t = \frac{p(H_t(\mathbf{x}) < \theta_t(z_t)|y = -1)}{p(H_t(\mathbf{x}) < \theta_t(z_t)|y = +1)} \tag{13}$$

satisfies $\hat{R}_t \geq \frac{1}{\alpha}$.

For the EnMS approach to be effective, the conditioning information z_t has to encode how well the other competing samples are classified and the function form of the threshold function $\theta_t(z_t)$ has to be simple enough to allow reliable estimation of its optimal parameters.

In our approach, the weak hypothesis h_t is evaluated for the whole set of competing samples $\mathcal{X}$ at a time, and the conditioning information is the maximum tentative classifier response on the competing samples

$$z_t = \max_{x \in \mathcal{X}}(H_t(x)). \tag{14}$$

We choose $\theta_t(z_t)$ as

$$\theta_t(z_t) = z_t - \lambda_t. \tag{15}$$

With this choice of $\theta_t(z_t)$, the EnMS condition for rejecting samples in Equation 12 becomes

$$H_t(x) < z_t - \lambda_t. \tag{16}$$

With these choices, EnMS introduces only a very small computational overhead. When computed sequentially, a weak hypothesis h_t can be computed on all active positions; then the maximal responses can be gathered and the samples fulfilling $H_t(x) < z_t - \lambda_t$ can be suppressed. When computing positions in parallel, the process has to be synchronized before the suppression step and gathering the maximal value may require synchronization, atomic instructions or a special value reduction method. However, even in highly parallel environments, the possible issues are not that significant as the potential serial operations are simple. Furthermore, suppression does not have to be performed after each weak hypothesis and the computation does not have to be strictly enforced without any significant performance drawbacks.

6. Runtime design

6.1 Exploiting SIMD architectures

The SIMD (Single Instruction Multiple Data) architectures exploit data level parallelism to accelerate certain operations. Contrary to instruction parallelism, the data parallelism is works so that the CPU performs the same instruction with vectors of data. This approach is very efficient in tasks where a simple computation is performed on large amount of data (e.g. stream processing).

Typically, CPUs contain a standard instruction set which processes integers and floats. This set is extended with a set of vector instructions which work over vectors of data stored in the memory. Vector instructions typically include standard arithmetic and logic instructions, instructions for data access and other data manipulation instructions (packing, unpacking, etc.). This is the case of general purpose CPUs like Intel, AMD or PowerPC. Beside the general purpose CPUs, there are GPGPU (General Purpose Graphics Processing Units), successors of

traditional GPUs (purposed to process graphics primitives) that can execute parallel kernels over data, and that can be viewed as advanced SIMD processors.

The SIMD architecture can be used especially to accelerate the following parts of detection.

- *Weak classifier evaluation* - the instructions can be used to evaluate multiple weak classifiers.
- *Feature evaluation* - the features like LRD, LRP and LBP can be evaluated in a data-parallel fashion.

When evaluating the weak hypotheses in a one-by-one manner, the evaluation of a feature can be transformed to SIMD processing so that all feature samples are loaded to registers and the response is evaluated by using SIMD instructions instead of a typical implementation by a loop [Herout et al. (2009); Juránek et al. (2010)]. This necessarily needs a pre-processing stage that transforms an image to a SIMD-friendly form and which allows for simple access to the data belonging to a feature - convolution of image. Speed up of this method compared to a naive implementation is very high, around 3 to 5, depending on the particular architecture on which it is implemented.

When evaluating multiple hypotheses, the implementations is pretty much the same as for one weak hypothesis without SIMD instructions. The difference is that the SIMD registers can hold information for more weak hypotheses (16 in the case of Intel SSE). This leads into efficient implementation of AdaBoost classifiers. WaldBoost classifiers, on the other hand, can be inefficient using this implementation as many weak hypotheses are calculated even when they are not necessarily needed for the classifier evaluation. Pre-processing is needed again to simplify the data access and feature evaluation. Speed-up achieved by this method is very high. In fact, when implementing WaldBoost evaluation, it is comparable to the method in the previous paragraph, even though many weak hypotheses are calculated unnecessarily.

In some cases, the feature response can be pre-calculated for all positions in the image and during detection, the feature is extracted by only one access to a pre-calculated image. In this case, for each version of a feature, an image with a pre-calculated result must be created. This is only possible when a small number of feature variant exist. For example, LBP with restricted size to 2×2 pixels ber block has four variants. On the other hand, LRD with the same restriction has 144 variants (as it is additionally parametrized by A and B indices) and calculation of such a high amount of images would be computationally expensive.

To summarize, benefits brought up by SIMD processing are the following: SIMD allows for features to be extracted very efficiently and the performance of a classifier evaluation can evan be increased by multiple number of times. On the other hand, the SIMD comes with the need of pre-processing which, when implemented without care, can reduce performance.

6.2 GPU implementation of the detection

Implementation of object detection in GPU was historically detected using programmable shaders [Polok et al. (2008)]; however, contemporary state of the art is in GP-GPU programming languages, such as CUDA or OpenCL [Herout et al. (2011)]. GP-GPUs programmed using one of these languages present one of the most powerful and efficient computational devices. When used for object detection, GP-GPUs can be seen as a SIMD device with a high level of parallelism.

Unfortunately, the high level of parallelism is difficult to employ in WaldBoost detection as the amount of computation in adjacent positions in the image is not correlated and in general is quite unpredictable, which fact heavily complicates usage of the ALUs in the SIMD device.

The efficient implementation of object detection using CUDA [Herout et al. (2011)] solves the problems of two main domains: the classifier operating on one fixed-size window, and parallel execution of this classifier on different locations of the input image. The problem of object detection by statistical classifiers can be divided into the following steps:

- loading and representing the classifier data
- image pre-processing
- classifier evaluation
- retrieving results.

The constant data containing the classifier (image features' parameters, prediction values of the weak hypotheses summed by the algorithm and WaldBoost thresholds) could be accommodated in the texture memory or constant memory of the CUDA architecture. These data are accessed in the evaluation of each feature at each position, so the demands for access speed are critical. Programs that are run on the graphics hardware using CUDA are executed as *kernels*; each kernel has a number of blocks and each block is further organized into threads. The code of the threads consumes hardware resources: registers and shared memory; this limits the number of threads that can be efficiently executed in a block (both the maximal and minimal number of threads).

One thread computes one or more locations of the scanning window in the image. The image pixels (or more precisely, window locations) are therefore divided into rectangular tiles, which are solved by different thread blocks. Experiments showed that the suitable number of threads per block was around 128. Executing blocks for only 128 pixels of the image would not be efficient, so we chose that one thread calculated more than one position of the window – a whole column of pixels in a rectangular tile. A good consequence of this layout is easy control of the resources used by one block: the number of threads is determined by the width of the tile, and the height controls the whole number of processed window positions by the block. The tile can extend over the whole height of the image or just a part of it. In order to avoid collisions of concurrently running threads and blocks, atomic increment (`atomicInc` function) of one shared word in the global memory is used for synchronization. This operation is rather costly, but the positive detections are so rare that this means of output can be afforded. As a consequence, the results of the whole process are at the end available in one spot of the global memory, which can be easily made available on the host computer.

The main property of the CUDA implementation is that the CUDA outperforms the CPU implementation mainly for high resolution videos. This can be explained by extra overhead connected with transferring the image to the GPU, starting the kernel programs, retrieving the results, etc. These overhead operations typically consume constant time independent of the problem size, so they are better amortized in high-resolution videos.

6.3 Programmable hardware

The runtime for object detection does not necessarily need to be implemented only in software; programmable hardware is one of the options as well, namely field programmable gate arrays

[Jin et al. (2009); Lai et al. (2007); Theocharides et al. (2006); Wei et al. (2004); Zemčík & Žádník (2007)]. While the algorithms of the object detection are in principle the same for software and hardware implementation, the hardware platform offers features largely different from the software and thus the optimal methods need to implement detection in programmable hardware are often different from the ones used in software and, in many cases, the hardware implementation may be very efficient.

The key features that are important for object detection are very different in hardware and software, and which are beneficial for hardware implementation, include:

- massive parallelism achievable with good performance/electrical power ratio
- variable data path width in hardware adjustable to exact algorithmic needs
- simple implementation of bit manipulation and logical functions
- nearly seamless complex control and data flow implementation

Of course, the hardware implementation also has severe limitations, the most important being:

- limited complexity of the hardware circuits
- computational resources for complex mathematical functions expensive
- memory structures relatively limited
- in most cases lower clock speed comparing to the processors

Taking into account the above advantages and limitations of programmable hardware, it can be considered for object detection designed specifically for the following cases:

- low end computational power embedded system with programmable hardware with programmable hardware as a co-processor; in this setup, it is expected that the programmable hardware performs more or less a complete detection task;
- high end computational system with programmable hardware as a pre-processing unit; this setup is different from the above one with respect to the detection which does not have to be done completely in programmable hardware, but rather the hardware is considered a resource to relieve the processor of the host system from as much computation as possible, and so it is feasible to implement perhaps incomplete but high performance pre-processor that reduces the need for computations;
- a complete object detection system in programmable hardware that can be combined with image pre-processing and where the complete detection task along with some image data flow considerations should be implemented.

Based on the above methods of exploitation, the methods of implementation of object detection in programmable hardware can be subdivided into a complete detection and pre-processing.

The typical methods of complete object detection in programmable hardware is feasible to implement using a sequential engine, possibly microprogrammable, which performs detection location by location, weak classifier by weak classifier until a decision is reached. As the evaluation of each weak classifier is relatively complex, the operation of the sequential unit is pipelined, so that several instances can be running in parallel. At the same time, different locations, in general, require a different number of weak classifiers to be evaluated. These

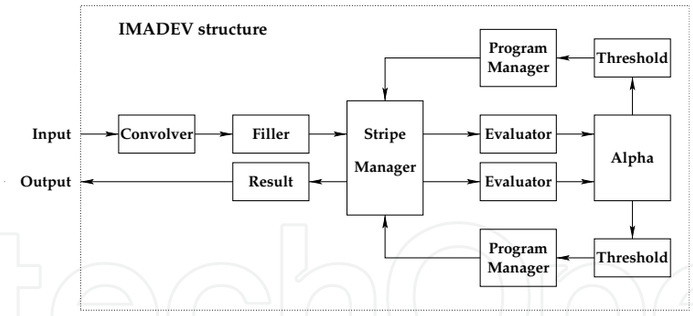

Fig. 6. Block structure of the object detector in programmable hardware (source Zemčík & Žádník (2007)).

facts lead into relatively complex timing and synchronization of processing; however, very good performance can be achieved [Zemčík & Žádník (2007)].

In a situation, where a complete evaluation of the detection is not required (e.g. in cases where a powerful CPU is available) and programmable hardware can be exploited for pre-processing, the best approach is probably a synthesis of fixed-function circuits synthesized based on results of the machine learning process "on demand" for each classifier. Such a synthesized circuit is most efficient when processing a (small) fixed number of weak classifiers for every evaluated position. While some of the weak classifiers are in such cases evaluated unnecessarily (assuming WaldBoost algorithm), the average price of weak classifier implementation is still often much lower than in the sequential machine described above. The main advantage of this approach is that all weak classifiers can be evaluated in a parallel way. However, as each weak classifier consumes chip resources, only a very small number of weak classifiers can be implemented in this way.

7. Results

7.1 Classifier cost minimization

This section gives an example of optimization of classifier performance by the balancing amount of computation between a fast hardware pre-processing unit and software post-processing unit. The classifiers used in this experiment were face detectors composed from 1000 weak hypotheses with LBP features and different false negative error rates (in a range from 0.02 to 0.2).

As a baseline, software implementation working on an integral image was selected, as it is the standard way of implementation of the detection. The other implementations ised in the experiments were SSE implementation that evaluate features one by one (SSE-A), and the SSE implementation that evaluates 16 weak hypotheses in a row (SSE-B).

The cost of the hardware unit was selected according to the area on the chip taken by the design. We set the cost constantly to $c_i = \frac{1}{m}$ where m is the maximal number of hypotheses that can be fit in the circuit. In this experiment, we use $m = 50$. In general, setting the cost to a low value, we simply say that the cost of the hardware unit is not of much interest to us, and conversely, setting the cost to a large value, we say that the cost of the hardware is very important. The cost of the post-processing unit was calculated from the measurement of

	Cost per weak hyp.
INTEGRAL (ref.)	0.215
SSE-A	0.110
SSE-B	0.070
FPGA	0.002

Table 1. Costs of weak hypotheses evaluation in different implementations of detection runtime used in the experiment.

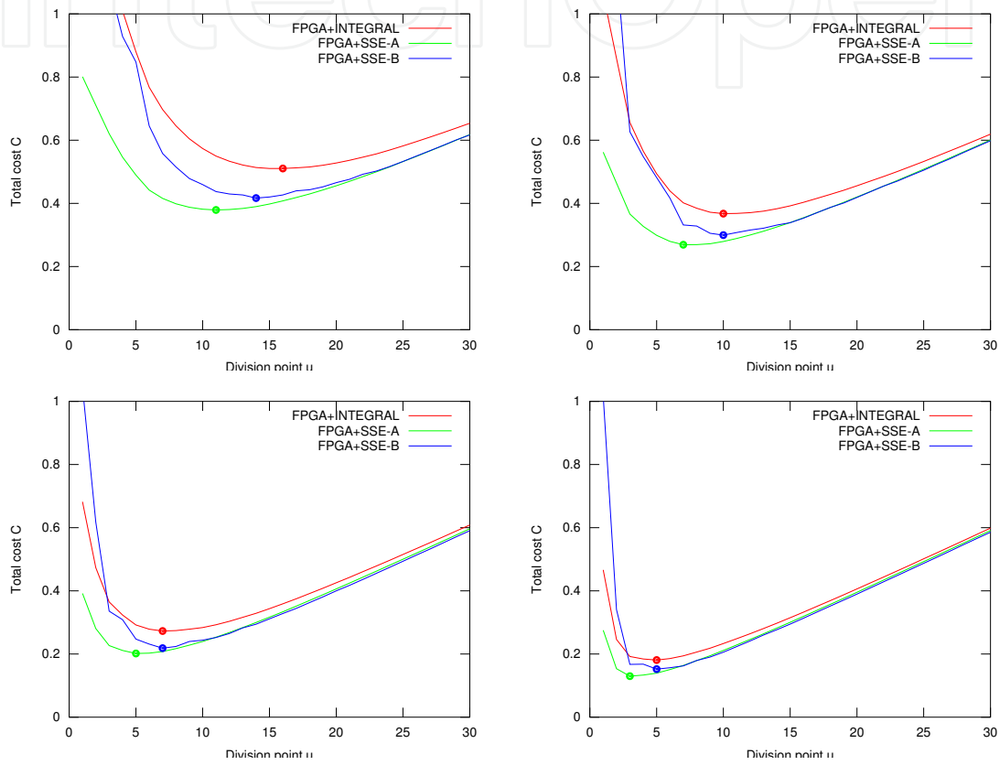

Fig. 7. Optimization results for classifiers with different false negative rates. Each plot shows the total cost of composition of FPGA with a software implementation. The division point is on the horizontal axis and the cost on the vertical axis.

processing time of the implementations of a standard PC, and it corresponds to microseconds per weak hypothesis. The cost values are summarized in Table 1. According to selected costs, the optimization minimize circuit area and, at the same time, the amount of computations in the software. By the combination of such diverse cost measures the result given by the optimization can be viewed as a "relative cost", but the interpretation of the value might be somewhat problematical. This does not, however, matter too much as we do not care about the absolute value of the cost, but about the position of the minima.

Figure 7 shows four plots of total cost for different classifiers. Each plot shows the value of total cost for different settings of the classifier division point and each curve corresponds to a particular combination of FPGA and software implementation. The results of optimization for a classifier with $\alpha = 0.02$ are summarized in Table 2. The *Division* column shows the division

	Division	Best cost	Computations
Integral	0/1000	1.56	0/1
SSE-A	0/1000	0.80	0/1
SSE-B	0/1000	1.24	0/1
FPGA+Integral	16/977	0.51	0.87/0.13
FPGA+SSE-A	11/988	0.38	0.78/0.22
FPGA+SSE-B	14/984	0.41	0.85/0.15

Table 2. Summary of results for classifier with LBP features and $\alpha = 0.02$.

of the classifier between hardware and software units; the *Best cost* column reflects the relative cost of the best solution and the *Computations* column shows the fraction of computations performed in hardware and software units.

This example shows that it can be beneficial to use a combination of more implementations of detection instead of one. It turns out that using a hardware pre-processing unit improves the detection performance (in terms of computational effort). Additionally, improving the performance of the software part allows for using shorter classifiers in hardware. This is an important fact as the FPGAs (and especially the cheaper ones) have typically limited resources and it could be impossible to put longer classifiers in them. Even higher performance could be achieved by using a neighborhood suppression method which would affect stage execution probability p in the optimization. This would result in shorter pre-processing units and lower total cost.

The application of such classifier optimization is, for example, in the field of smart camera design. The pre-processing module can be placed directly in the camera which then outputs, beside the normal image, the image with potential occurrence of target objects. Such information, as the above example has shown, dramatically decreases the required computation time in the post-processing module.

7.2 Neighborhood suppression results

The suppression of neighboring positions was tested on the standard frontal face MIT+CMU dataset. Three WaldBoost classifiers with target false positive rates of 0.01, 0.05 and 0.2 were trained for four types of image features: LRD, LRP, LBP and Haar. For each classifier, three neighborhood suppression strategies were trained with target false positive rates of 0.01, 0.05 and 0.2. Comparing results of the combinations allows us to evaluate if it is more effective to use neighborhood suppression than just by using a WaldBoost classifier with a higher false positive rate. The results of this experiment in Fig. 8 clearly show that neighborhood suppression is indeed effective and on average it evaluates less weak hypotheses per image position for the same accuracy.

7.3 EnMS results

EnMS was evaluated on a face localization task. The dataset was downloaded from Flicker groups *portraits* (training) and *just_faces* (testing). The dataset contains $84,251$ training and $6,704$ near-frontal faces. The images were rescaled to a 100×100 pixel resolution with the face approximately 50×50 pixels large and positioned in the middle. Both WaldBoost and

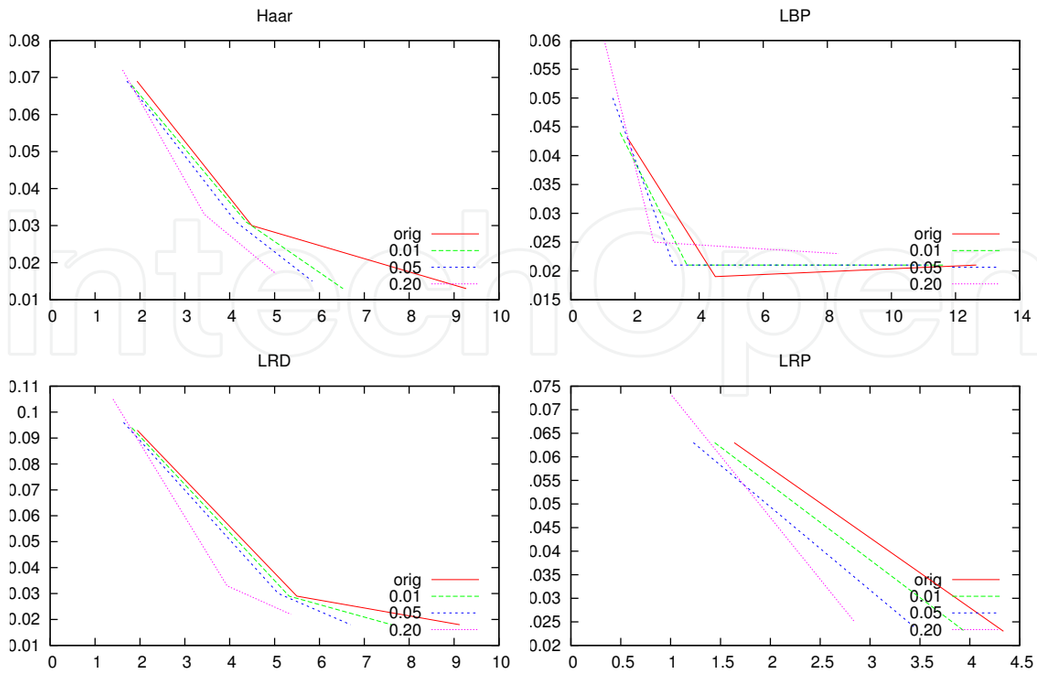

Fig. 8. The graphs show AUC on y-axis (Area Under ROC) versus the average number of weak classifiers evaluated per image position as measured on the MIT+CMU frontal face dataset. The individual lines are for original WaldBoost detectors without neighborhood suppression (full line) and the other lines are with added neighborhood suppression with different target false negative rates. Good results should be in the left (fast) bottom (accurate) corner.

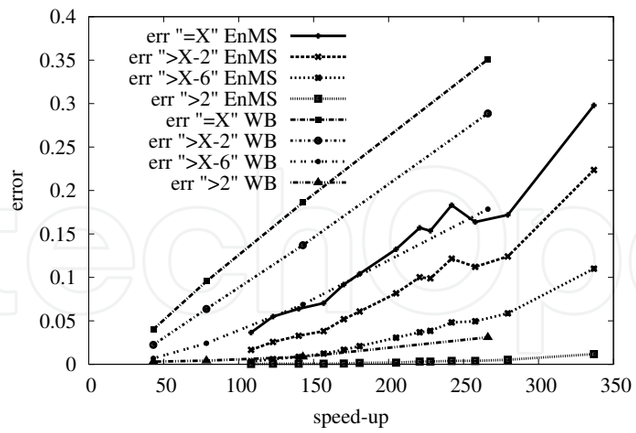

Fig. 9. The graphs show frontal face localization error (y-axis) for different speed-ups achieved by WaldBoost and EnMS. The speed-up is measured as reduction of the number of weak hypotheses evaluated on average per image position relative to the full length of the classifier (length is the same for WaldBoost and EnMS). The lines represent differently computed errors (see text) of WaldBoost and EnMS.

EnMS were evaluated on this data for several target false negative rates. The localization accuracy was measured as the number of images where the detector returned a position with the highest response of a classifier which always evaluated all weak hypotheses. In order to allow for some tolerance, errors were also counted as failure to detect position with the reference classifier response lower by 2 and 6 than the best response and failure to detect position with the reference response higher than 2 which is an operating point that still gives reasonably low false alarms in the detection task. The results in Fig. 9 show that EnMS provides approximately two times better speed for the same error rates than WaldBoost.

8. Conclusions

This chapter focused on methods of real-time object detection with classifiers. It has been demonstrated that the object detection methods working in real-time are feasible and can be implemented on a variety of platforms, such as personal computer processors, GP-GPU platforms, or even in programmable hardware.

In order to achieve real-time performance, an efficient implementation platform and efficient implementation itself is necessary, but further enhancement through algorithmic acceleration is needed as well. Two examples of such acceleration are presented in the chapter: exploitation of information about neighborhoods of the already classified positions in the image and early suppression of non-maxima of the classifier responses. The approach of exploitation of the neighborhoods in the image is based on the idea that classification of the overlapping sub-images in the image - the neighborhoods - may share some properties and information. One of the possible ways to share such information is through re-using the weak classifiers used during classification of one location through WaldBoost for predicting results in the other neighboring locations. This prediction is done through a machine learning process similar to WaldBoost where the difference to WaldBoost is that the training process actually reuses the already selected weak classifiers that were used at the original location. While this process works well only in close neighborhoods, it brings a significant speed-up.

Pre-processing that rules out some parts of the image from the detection process can significantly speed up the detection process. Important future research certainly includes machine-learning based pre-processing methods and research of under-sampling in scanning methods that can also improve detection performance possibly without any adverse effects on precision. Future research also includes algorithmic improvements of acceleration methods, such as improvement in the processor assignment in GP-GPU, improved scanning trajectories in neighborhood exploitation, or further improvements in feature extraction.

9. Acknowledgement

This work has been supported by the FIT VUT Brno project "Advanced recognition and presentation of multimedia data", FIT VUT, FIT-S-11-2, Centre of excellence in computer science "The IT4Innovations Centre of Excellence", EU, CZ 1.05/1.1.00/02.0070, "Reduced Certification Costs Using Trusted Multi-core Platforms", Artemis JU, RECOMP #100202, "Smart Multicore Embedded Systems", Artemis JU, SMECY #100230, and the Czech Ministry of Education, Youth and Sports, "Security-Oriented Research in Information Technology", CEZ MŠMT, MSM0021630528 and “Centre of Computer Graphics", MŠMT, LC06008.

10. References

Bay, H., Ess, A., Tuytelaars, T. & Van Gool, L. (2008). Speeded-up robust features (surf), *Comput. Vis. Image Underst.* 110(3): 346–359.

Blaschko, M. B. & Lampert, C. H. (2009). Object localization with global and local context kernels, *British Machine Vision Conference*.

Bourdev, L. & Brandt, J. (2005). Robust object detection via soft cascade, *CVPR*.

Chen, J., Shan, S., Yang, P., Yan, S., Chen, X. & Gao, W. (2004). Novel face detection method based on gabor features, *Sinobiometrics 2004*, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, pp. 90–99.

Dalal, N. & Triggs, B. (2005). Histograms of oriented gradients for human detection, *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) – Volume 1*, IEEE Computer Society, Washington, DC, USA, pp. 886–893.

Fidler, S. & Leonardis, A. (2007). Towards scalable representations of object categories: Learning a hierarchy of parts, *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on* 0: 1–8.

Freund, Y. (1995). Boosting a weak learning algorithm by majority, *Inf. Comput.* 121(2): 256–285.

Friedman, J., Hastie, T. & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting, *Annals of Statistics* 28: 2000.

Grove, A. J. & Schuurmans, D. (1998). Boosting in the limit: Maximizing the margin of learned ensembles, *In Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 692–699.

Herout, A., Hradiš, M. & Zemčík, P. (2011). Enms: Early non-maxima suppression, *Pattern Analysis and Applications* 2011(1111): 10.

Herout, A., Zemčík, P., Hradiš, M., Juránek, R., Havel, J., Jošth, R. & Žádník, M. (2009). *Low-Level Image Features for Real-Time Object Detection*, IN-TECH Education and Publishing, p. 25.

Hradiš, M., Herout, A. & Zemčík, P. (2008). Local rank patterns - novel features for rapid object detection, *Proceedings of International Conference on Computer Vision and Graphics 2008*, Lecture Notes in Computer Science, pp. 1–2.

Jin, S., Kim, D., Nguyen, T. T., Jun, B., Kim, D. & Jeon, J. W. (2009). An fpga-based parallel hardware architecture for real-time face detection using a face certainty map, *Application-Specific Systems, Architectures and Processors, IEEE International Conference on* 0: 61–66.

Juránek, R., Herout, A. & Zemčík, P. (2010). Implementing local binary patterns with simd instructions of cpu, *Proceedings of Winter Seminar on Computer Graphics*, West Bohemian University, p. 5.

Lai, H.-C., Savvides, M. & Chen, T. (2007). Proposed fpga hardware architecture for high frame rate face detection using feature cascade classifiers, *First IEEE International Conference on Biometrics: Theory, Applications, and Systems, 2007. BTAS 2007.*, pp. 1 –6.

Lee, T. S. (1996). Image representation using 2d gabor wavelets, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18(10): 959–971.

Leibe, B., Leonardis, A. & Schiele, B. (2008). Robust object detection with interleaved categorization and segmentation, *Int. J. Comput. Vision* 77(1-3): 259–289.

Li, S., Zhang, Z., Shum, H. & Zhang, H. (2002). Floatboost learning for classification, *The Conference on Advances in Neural Information Processing Systems (NIPS)*.

Lowe, D. G. (1999). Object recognition from local scale-invariant features, *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, IEEE Computer Society, Washington, DC, USA, p. 1150.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints, *International Journal of Computer Vision* 60(2): 91–110.

Polok, L., Herout, A., Zemčík, P., Hradiš, M., Juránek, R. & Jošth, R. (2008). "local rank differences" image feature implemented on gpu, *Proceedings of the 10th International Conference on Advanced Concepts for Intelligent Vision Systems*, Lecture Notes In Computer Science; Vol. 5259, Springer Verlag, pp. 170–181.

Ratsch, G. (2001). *Robust Boosting via Convex Optimization: Theory and Applications*, PhD thesis, Mathematisch-Naturwissenschaftichen Fakultat der Universitat Potsdam.

Rudin, C., Schapire, R. E. & Daubechies, I. (2004). Boosting based on a smooth margin, *Learning Theory*, Vol. 3120/2004 of *Lecture Notes in Computer Science*, Springer, pp. 502–517.

Schapire, R. E., Freund, Y., Bartlett, P. & Lee, W. S. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods, *The Annals of Statistics* 26(5): 1651–1686.

Schapire, R. E. & Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions, *Mach. Learn.* 37(3): 297–336.

Serre, T., Wolf, L. & Poggio, T. (2005). Object recognition with features inspired by visual cortex, *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, IEEE Computer Society, Washington, DC, USA, pp. 994–1000.

Sochman, J. & Matas, J. (2004). Adaboost with totally corrective updates for fast face detection, *FGR*, pp. 445–450.

Sochman, J. & Matas, J. (2005). Waldboost - learning for time constrained sequential detection, *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, IEEE Computer Society, Washington, DC, USA, pp. 150–156.

Šochman, J. & Matas, J. (2007). Learning a fast emulator of a binary decision process, *in* Y. Yagi, S. B. Kang, I. S. Kweon & H. Zha (eds), *ACCV*, Vol. II of *LNSC*, Springer, Berlin Heidelberg, pp. 236–245.

Sochman, J. & Matas, J. (2009). Learning fast emulators of binary decision processes, *International Journal of Computer Vision* 83(2): 149–163.

Theocharides, T., Vijaykrishnan, N. & Irwin, M. (2006). A parallel architecture for hardware face detection, *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*.

Viola, P. & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 1: 511–518.

Wald, A. (1947). *Sequential Analysis*, John Wiley and Sons, Inc.

Wei, Y., Bing, X. & Chareonsak, C. (2004). Fpga implementation of adaboost algorithm for detection of face biometrics, *IEEE International Workshop on Biomedical Circuits and Systems*, pp. S1/6–17–20.

Zemcik, P., Hradis, M. & Herout, A. (2007). Local rank differences - novel features for image, *Proceedings of SCCG 2007*, pp. 1–12.

Zemčík, P. & Žádník, M. (2007). Adaboost engine, *Proceedings of FPL 2007*, IEEE Computer Society, p. 5.

Zemčík, P., Hradiš, M. & Herout, A. (2010). Exploiting neighbors for faster scanning window detection in images, *ACIVS 2010*, LNCS 6475, Springer Verlag, p. 12.

Zhang, L., Chu, R., Xiang, S., Liao, S. & Li, S. Z. (2007). Face detection based on multi-block lbp representation, *ICB*, pp. 11–18.

Real-Time Systems, Architecture, Scheduling, and Application
Edited by Dr. Seyed Morteza Babamir

ISBN 978-953-51-0510-7
Hard cover, 334 pages
Publisher InTech
Published online 11, April, 2012
Published in print edition April, 2012

This book is a rich text for introducing diverse aspects of real-time systems including architecture, specification and verification, scheduling and real world applications. It is useful for advanced graduate students and researchers in a wide range of disciplines impacted by embedded computing and software. Since the book covers the most recent advances in real-time systems and communications networks, it serves as a vehicle for technology transition within the real-time systems community of systems architects, designers, technologists, and system analysts. Real-time applications are used in daily operations, such as engine and break mechanisms in cars, traffic light and air-traffic control and heart beat and blood pressure monitoring. This book includes 15 chapters arranged in 4 sections, Architecture (chapters 1-4), Specification and Verification (chapters 5-6), Scheduling (chapters 7-9) and Real word applications (chapters 10-15).

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Roman Juranek, Pavel Zemˇcík and Michal Hradis (2012). Real-Time Algorithms of Object Detection Using Classifiers, Real-Time Systems, Architecture, Scheduling, and Application, Dr. Seyed Morteza Babamir (Ed.), ISBN: 978-953-51-0510-7, InTech, Available from: http://www.intechopen.com/books/real-time-systems-architecture-scheduling-and-application/real-time-algorithms-of-object-detection-with-classifiers

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen