Acceleration of Object Detection using Classifiers

Roman Juránek*
Department of Computer Graphics and Multimedia
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, 612 00 Brno, Czech Republic
ijuranek@fit.vutbr.cz

Abstract

Detection of objects in computer vision is a complex task. One of the most popular and well explored approaches is the use of statistical classifiers and scanning windows. In this approach, classifiers learned by AdaBoost algorithm are often used as they achieve low error rates and high detection rates. To achieve high performance, acceleration techniques are used, such as use of GPU, SIMD, custom hardware etc. The contribution of this thesis is introduction of a technique which enhance object detection performance with respect to an user defined cost function. The presented method balances computations of previously learned classifier between two or more run-time implementations in order to minimize the cost function. The optimization method is verified on a basic example – division of classifier to a pre-processing unit implemented in FPGA, and a post-processing unit in a standard PC.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Object Detection, AdaBoost, WaldBoost, Acceleration, SIMD, Cost Minimization

1. Introduction

Many real life applications could use information about objects captured by a camera. In user interfaces, for example, a camera can be used as an alternative input device. The computer can capture the scene in front of the computer and analyze it in order to find user's face and Analise the user's gestures. Such input can be used to control the computer without the keyboard or mouse. Other applications, like traffic control or surveillance systems, use detection of objects to automatically count people, check license plates or record traffic violations. Object detection is important part of such systems.

Very popular approach to object detection is exploitation of statistical classifiers and scanning windows technique. The classifier is learned by a machine learning algorithm, typically supervised or semi-supervised. Among the large number of statistical machine learning methods, most prominent in real-time object detection is the Adaptive Boosting algorithm and its modifications. This method became so popular (due to its simplicity and performance) that it found its way to commercial systems that use object detection.

The focus in this paper is on *implementational* acceleration of the detection process. The contribution is the introduction of *the technique for composition of different implementations of object detection in order to enhance its performance with respect to an user defined cost function.* The composition balances computations between two or more implementations in order to minimize the cost function. The method is based on the analysis of previously learned classifier and knowledge of properties of the implementations which executes the detection. The method is verified on the example where a classifier is divided to the two evaluation phases, first executed in a hypothetical hardware unit and the second executed in software using implementation with different properties. The classifiers in the thesis are learned by the WaldBoost [23] algorithm and they are frontal face detectors. Image features used are LBP (Local Binary Patterns) [30] and LRF (Local Rank Functions) [11]. This combination of learning algorithm and image features can be considered as a state of the art in real-time object detection with scanning-window classifiers.

The rest of this paper is structured as follows. Section 2 briefly summarizes detection of objects using WaldBoost classifiers and methods for acceleration of the detection. Section 3 introduces the contribution of the thesis – evaluation of classification cost and its minimization using different implementations of the detection. The experimental results are presented in Section 4 along with the discussion and the paper is concluded in Section 5.

2. Object Detection using Classifiers

2.1 The Detection

The basic part of object detection with classifiers is the classifier which is learned by a machine learning algorithm [13, 5, 25, 23, 1]. The classifier can decide if the input

*Recommended by thesis supervisor: Doc. Dr. Ing. Pavel Zemčík Defended at Faculty of Information Technology, Brno University of Technology, Brno.

sample image is or is not an object of interest. Object detection with classifiers is achieved by analysis of each sub-window of input image by the classifier. The sub-windows are taken from all positions and scales (and possibly other transformation, depending on the particular application). The number of analyzed samples is very high, reaching hundreds of thousands per image and thus the resulting classifier should have very low *false alarm rate*. On the other hand, each object appears in multiple neighboring sub-windows and thus *false negative rate* does not necessarily need to be zero and some detections can be, in fact, missed. It is only required that at least one of these positive sub-windows are hit by the classifier. The detection with sliding windows is exploited in many works. In the [19] they use over-complete set of Haar-like features and SVM classifier as basis for general object detection framework. Similar approach was used in [25] where they use AdaBoost. In [4] authors use histograms of oriented gradients as an input to SVM classifier to detect pedestrians. In [24] the authors propose emulator of key point detectors using sliding window detector based on WaldBoost algorithm.

The WaldBoost algorithm [23] uses Real AdaBoost [22] to select and order weak classifiers $h(x)$ (simple decision functions at least slightly better than random decision) and adds Wald's *Sequential Probability Ratio Test* [26]. WaldBoost produces a soft-cascade [2, 27, 23] classifiers with near optimal decision strategy which minimizes average number of weak classifiers evaluated per sample for target *false negative rate* α.

$$\begin{aligned} H_t(x) &= \sum_{i=1}^{t} h_i(x) \\ S_t &= \begin{cases} -1 & \text{when } H_t(x) > \theta_t \\ & \text{Otherwise evaluate } S_{t+1} \end{cases} \end{aligned} \quad (1)$$

The evaluation of a classifier $H(x)$ for input sample x is defined by Equation 1. From the implementation point of view, given the sample x, the classifier sequentially evaluates functions $h_t(x)$, $t \in 0, 1.., T$, and accumulates the strong classifier response H_t. In each step t the H_t is compared to θ_t and the sample is rejected when $H_t(x) < \theta_t$, otherwise the evaluation proceeds with the step $t + 1$.

In this work, WaldBoost is used for classifier learning and Local Binary Patterns (LBP) [30] and Local Rank Functions (LRD, LRP) [11] are used as a basis for weak classifiers.

2.2 Acceleration Methods

The acceleration can be done in several ways. On the algorithmic level, the speed-up can be achieved by modification of classifier structure, e.g. Cascade [25] or soft-cascade [2, 23] is more optimal than pure AdaBoost. Other modification is that the classifier can use an information from a sub-window to predict responses on neighboring positions of sub-windows [28]. Other approaches are possible as well [8]. Beside the algorithmic acceleration, the implementation can be fine-tuned to better exploit the underlying computational architecture, e.g. multi-core architectures, SIMD, graphics hardware and custom hardware.

On the multi-core architectures, process can be parallelized by the control-flow transformation employing more computing elements. In practice, it means to use multi-thread interfaces (like threads, OpenMP [3], Intel Thread Building Blocks (TBB) [21], Posix threads and others) to execute a code on the code. In the context of object detection, the multiple computing cores can be used for *processing of multiple frames*, *processing of multiple sub-windows* and *processing of multiple weak classifiers in the same sub-window*.

Different acceleration possibility is to use properties of SIMD architectures offering data-parallel processing. Many modern CPUs (like Intel, AMD or PowerPC) contain standard instruction set which can process integers and floats. This set is extended with a set of vector instructions which work over vectors of data stored in registers. Vector instructions typically include standard arithmetic and logic instructions, instructions for data access and other data manipulation instructions (load/store, packing, unpacking, etc.). The SIMD principles can be used with great benefit in applications where large data is processed; e.g. image processing (FIR filtering). Each instruction can load a vector of data and apply an operation on them. Then, for example, memory copying can be accelerated by moving blocks of data (e.g. 16 byte blocks using Intel SSE) instead of individual bytes. On the other hand, not all algorithms can be vectorized as the flow control can depend on the data. In the context of the object detection, the main application of the data parallelism is in feature extraction [7, 16]. There are two main options how to employ it. First, to keep the evaluation code *as is* and calculate N spatially close responses simultaneously. The second method is to calculate only a single feature at the time and process *all feature data* in parallel using SIMD instructions [16].

Graphics hardware offers interesting tools for implementation of the detection as well. Programmable shaders can be used on traditional GPUs (Graphics Processing Units) [20] through OpenGL or DirectX interfaces using shader programming languages (GLSL or HLSL). GPGPUs (General Purpose GPUs) can be used through CUDA or OpenCL programming languages [10, 9].

Programmable hardware, namely Field Programmable Gate Arrays (FPGA) can be used for the detection as well [29, 6, 12, 15]. While the algorithms of the object detection are in the principle the same for software and hardware, the hardware platform offers features largely different from the software and thus the optimal methods to implement detection in programmable hardware are often different from the ones used in software. In many cases, the hardware implementation can be more efficient than the software implementation. Typical use of hardware is a) preprocessing where an image is processed in order to reject majority of background samples [18], and b) complete detection where the hardware unit outputs parameters of the detected objects [29].

3. Classification Cost and its Minimization

This section presents contribution of the thesis – *minimization of classification cost of WaldBoost classifiers through combination of different run-time implementations of object detection.* Sources of the classification cost are twofold. Firstly, the classifier has its inherent cost given by the learning process. When executing the classifica-

tion on a set of images, there is average number of weak classifiers that has to be evaluated to reach the decision. Classifiers executing lower number of weak classifiers are faster and have therefore lower cost. Secondly, the cost is a property of the classification engine (or run-time implementation) in which the object detection is executed. The engine can be implemented by various methods – parallel evaluation of weak classifiers, parallel evaluation of image sub-windows, etc.; and on various platforms offering different means of classifier evaluation – SIMD, FPGA, etc. The knowledge of the classifier properties and properties of the detection engines can be used for reduction of computational effort. The reduction can be performed by combining two or more detection engines, each executing different part of classifier [17]; e.g.: hardware preprocessing unit connected to post-processing unit on traditional CPU. The reduction can be applied to various types of cost (computations, memory, hardware price, etc.). In this thesis, the interest is in minimization of computational effort and the relative cost thus roughly corresponds to the computational time (except where noted otherwise).

3.1 Classifier Properties

The main property of WaldBoost classifiers is the probability of evaluation of a weak classifier, reflecting how often a weak classifier 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 rejection nature of WaldBoost classifiers, the sequence of p_i is decreasing. The first stage is evaluated always; i.e. the $p_0 = 1$. The evaluation probability captures *intrinsic* computational complexity of the classifier.

The stage execution probability depends mainly on the classifier rejection rate – how rapidly are samples rejected by the early weak classifiers. A classifier, can have different stage execution probabilities when using different implementations of the evaluation. For example, consider an implementation which evaluates four weak classifiers in one step and applies the WaldBoost threshold after this 'bunch' is evaluated. First four weak classifiers would have probability of evaluation equal to 1, even though execution of all of them is not necessary in most cases. The next four would have probability equal to each other, and so on. Therefore, the stage execution probability is a property of the classifier *and* the implementation of its evaluation.

3.2 Cost Evaluation

In the case of the AdaBoost and WaldBoost classifiers the total cost C is proportional to sum of individual costs of executed weak classifiers and it can be calculated by (2). The T is the length of classifier, k is the overall classifier cost which symbolizes evaluation cost on particular platform on which the classification is implemented. The p is probability of execution of particular weak classifier (see Section 3.1). The c is relative cost of the weak classifier evaluation which addresses the possibility that the weak classifiers have different cost (due to use of different features for example).

$$C = k \sum_{i=1}^{T} p_i c_i \tag{2}$$

When analyzing real classifiers, p can be obtained from the statistics on input images and can be obtained c by time measurement or other cost estimation and k can be set to a constant value ($k = 1$). In the object detection, most common are homogeneous classifiers; i.e. those with all weak classifiers of the same type and with same type of features. In such cases, the cost of weak classifier is constant $c_i = c$. Additionally in AdaBoost, all weak classifiers are executed every time and the probability of execution of all weak classifiers is equal to $p_i = 1$. The C from (2) can be thus simplified to $C^{(AB)}$ (for AdaBoost) and $C^{(WB)}$ (for WaldBoost) in (3).

$$C^{(AB)} = knc \qquad C^{(WB)} = kc \sum_{i=1}^{n} p_i \tag{3}$$

Considering a classifier of length T, the value C gives us an expected cost of evaluation of the classifier. The measure is abstract and it can express different facts about the analyzed classifier. For example, when the c is set to 1, the measure express number of weak classifiers executed in average. Or when set according to time needed to evaluate particular classifier, the C express average time needed to evaluate the classifier.

3.3 Cost Minimization

Besides the properties of a classifier, the properties of runtime implementation also contributes to the total cost. Implementations with different properties exist – differences can be in design of feature extraction, image scan, multi-scale detection, etc. Imagine, for example, an implementation A which can very efficiently evaluate $K > 1$ weak classifiers in a row, but it always evaluates *all* of them no matter how many weak classifiers is actually needed for the evaluation. It could be pre-processing unit implemented in hardware which rejects areas without occurrence of target object. The other implementation B in software can evaluate the classifier in standard way. The computational cost for one feature in A is much lower than in B but implementing the whole classifier in the hardware is hard to achieve due to limited resources. Moreover it could be uneconomic to do so as the hardware resources are relatively expensive.

$$C = \underset{0 \le u \le T}{\arg\min} \left(k_1 \sum_{i=0}^{u-1} p_{1,i} c_{1,i} + k_2 \sum_{i=u}^{T-1} p_{2,i} c_{2,i} \right) \tag{4}$$

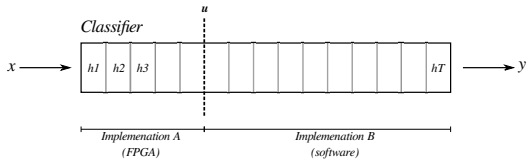

Figure 1: Composition of two implementations of classification. First u classifiers are evaluated in FPGA and the rest in software.

Both implementations can be put together in a composed implementation. The problem here is how many weak classifiers should be put in hardware unit and how many is left for the software. The cost of both parts can be

measured and sum of the individual costs of both parts gives total cost C. The composition with the minimal total cost can be found using Equation (4).

The composition of two phases can be fine tuned by one parameter – division point u. Equation 4 shows the minimization problem where C is total minimal cost of the evaluation, u is point of classifier division, k, c and p correspond to the parameters of the cost computation from Equation 2. Note, please, that although the properties p of classifier are same for both parts, the p can be in general different for each part. This is due to the structure of the evaluation in the particular implementation which can *force* different probabilities of feature evaluation e.g. by evaluating more features in one step (see Section 3.1).

When going beyond the example given above, more than two phases of evaluation and the minimization is thus multi-dimensional. In general case described by (5), the classifier division is vector $\mathbf{u}$ which values are searched for in order to find best composition of parts with different properties. Note that u_i can be equal to u_{i+1} and some part could be in fact skipped when it is evaluated as useless in the optimization.

$$
\begin{aligned}
C = \arg\min_{\mathbf{u}} & \sum_{m=1}^{M} \left(k_m \sum_{i=u_{m-1}}^{u_m - 1} p_{m,i} c_{m,i} \right) \\
& s.t. \\
& u_0 = 0 \\
& u_M = T \\
& u_{i-1} \leq u_i, \quad 0 \leq i \leq M
\end{aligned}
\tag{5}
$$

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

4. Experiments and Results

The objective of the experiments is to test the hypothesis about minimization of total cost using composition of classifier run-time implementations. Firstly, the implementations used in this thesis are described and their performance is measured. This measurement serves as cost measure in the subsequent experiments. Secondly, classification cost of different composition is minimized using principles described in Section 3 and the theoretical calculations are compared to the real measurements. The evaluation metric used in the thesis is improvement of performance of object detection.

4.1 Classification Cost Measurements

Classifiers used in the experiments were face detectors trained with WaldBoost algorithm in experimental framework developed at Faculty of Information Technology [14]. Classifiers with different properties were used in the experiments – LBP features, $\alpha \in \{0.02, 0.05, 0.1, 0.2\}$ and maximum 2×2 pixel block.

As a baseline, software implementation working on integral image was selected, as it is a standard way of implementation of the detection. The other used implementations in software were implementations that use SSE instructions either for evaluation of features (SSE-A, SSE-B) or pre-processing (SSE-C).

	PC1
CPU	Intel Core i5
Cores	4
Frequency	3.3 GHz
Memory	4 GB
OS	Debian 32 bit

Table 1: Specifications of the two machines used in the experiments.

Additionally, an FPGA pre-processing unit (FPGA) is used in the experiments [18]. This unit can contain up to N classifiers and it evaluates always *all* weak classifiers in parallel and thus it works as a simple AdaBoost unit which applies WaldBoost thresholds after evaluation of all weak hypotheses.

The cost of software implementations correspond to a time needed for evaluation of a weak classifier. The time was measured on a dataset of 130 images (CMU dataset) with different resolution and content. Each image was processed ten times and the times were averaged. In total, the cost was evaluated from around 26 million classifications which is enough to get accurate estimate. The measurement was performed on two different computers (Table 1) with CPU frequency set to constant value; i.e. no frequency scaling allowed, and single-thread code was used.

The cost of the classifier in the hardware implementation was set as an area needed for the classifier in the FPGA circuit, reflecting the cost of the chip. In this experiments, the cost is set constantly to $c_i = \frac{1}{N}$ where N corresponds to a number of weak classifiers which can be efficiently stored in typical low cost FPGA ($N = 50$ is assumed in experiments). Certainly, better cost functions could be found. For example cost incorporating properties of real device, like feature evaluation speed or or real price of the device. The cost selected in this work is selected to illustrate the principle of the cost minimization. In general, setting the cots 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.

For each classifier, stage execution probability (see Section 3.1) was obtained from detection results on the CMU dataset.

Table 2 shows measurements of classification cost c_i for different types of features for different implementation. The cost value is *independent* on the value of classifier α. The costs for all classifiers were thus averaged to get more precise estimate.

On the left side, Fig. 2 shows stage execution probabilities p_i for the classifiers in the experiments. Note that p_i for classifiers with higher false negative rate α decrease faster. This results in lower number of weak classifiers evaluated in average, and ultimately, to higher classification speed compared to the more accurate classifiers with lower false

	PC1		
	LBP	LRD	LRP
INTEGRAL (ref.)	0.0421	0.0466	0.0464
SSE-A	0.0236	0.0269	0.0306
SSE-B	0.0114	0.0129	0.0117
SSE-C	0.015	–	–

Table 2: Costs c_i of weak hypotheses evaluation in different implementations of detection run-time on two different computers used in the experiments. Note that the SSE-C implementation can evaluate only LBP features.

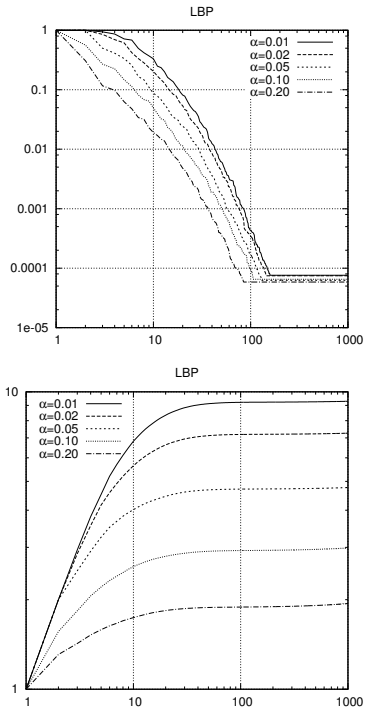

Figure 2: Top, stage execution probabilities for classifiers used in the experiments. Bottom, classifier cost (measured in number of weak classifiers) grows with length of the classifier.

negative rate. The right column of Fig. 2 shows speed comparison of the classifiers – average number of weak classifiers evaluated per window.

4.2 Cost Minimization

This section gives results of classification cost minimization based on results described in previous section. First, compositions of classifier implementations used in the experiments are described. Then the results are presented and compared to real measurements.

In this experiment, classifiers were divided in two parts. First part is evaluated in the hardware pre-processing unit. The second part is left for evaluation in the software using one of the above described run-time implementations. Classifier cost estimation method and cost minimization described in Section 3 is used to estimate optimal length of the pre-processing unit according to the cost measure defined in Section 4.1. The optimization objective is thus minimization of circuit area and, at the same time, minimization of the amount of computations in the software. By combination of such diverse cost measures the result (total cost C) given by the cost evaluation 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 value of the cost but about the position of the minima.

Figure 3 shows results of the minimization of total cost on the PC1 for the classifiers described above. The plots show dependence of total cost on setting of classifier division point. The division with minimal cost is marked by a circle. Note that slower classifiers (low false negative rates, α) result in longer part in hardware unit, meaning that the hardware should take care of majority of computations and the post-processing is left for a software. Another notable fact is that when using slower implementation in the software (or slower computer), larger part of computations is left for hardware unit. This means that faster computers/implementations tend to compute more weak classifiers in software.

Comparison of the cost minimization with real measurement in a few selected cases is shown in Fig. 4. Note that although the scale of the curves can be sometimes different, the position of the minima is approximately in the place predicted by the optimization. The difference is caused mainly by the overhead introduced by switching of run-time during the detection, data and instruction caching, etc. (which is not included in the optimization). The optimization thus gives a good advice for classifier division.

5. Conclusion

This paper presented a method for calculation of classification cost for WaldBoost classifiers and a method for composition of different run-time implementation into a coherent unit with minimized cost. In the experiments, presented in Section 4, the hypothesis about the composition of of the run-time implementations has been verified on an example with a hypothetical hardware unit and a software implementation. It turns out that it is indeed beneficial to divide the classifier evaluation into two (or even more) parts. This division allows for moving significant portion of computations into the implementation which can efficiently reject most of background subwindows and leave the rest of computations for potentially slower software unit. These results are supported by measurements The optimization of the division can be tuned for the particular classifier and particular hardware on which the detection is executed.

The principles described in this thesis can be used, for example, to automatically tune object detection for best performance with respect to the properties of the classifier and the machine on which the detection is executed, by combination of more implementations of the classifier evaluation. Other example can be a design of smart camera which can produce, along with standard image, an image pre-processed by a classifier or directly parameters of detected objects. The cost minimization allows for composition of parallel and sequential hardware units and the cost criterion used for minimization can be e.g. power consumption, chip area or other.

Further research include implementation of the object de-

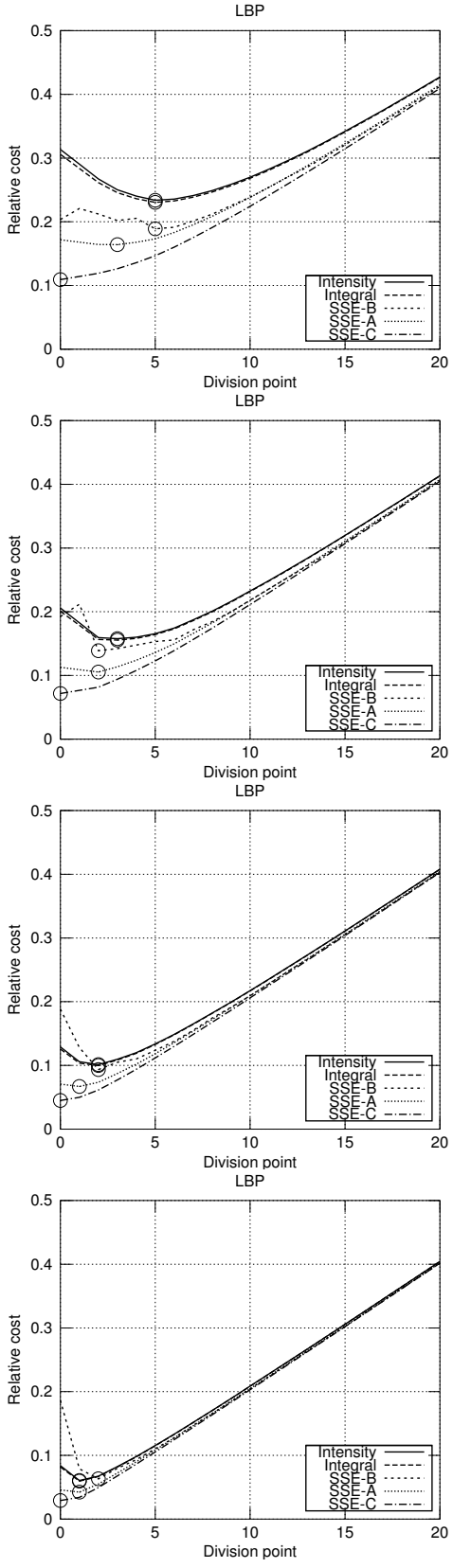

Figure 3: Search for optimal division point on PC1. Classifiers ware divided to two parts, first executed in FPGA and the second in PC1 (different run-times are showed as curves in plots). The plots shows the cost evaluation for four different classifiers with α of 0.02, 0.05, 0.1 and 0.2 respectively. LBP features were used. The position with minimal cost is marked by a point.

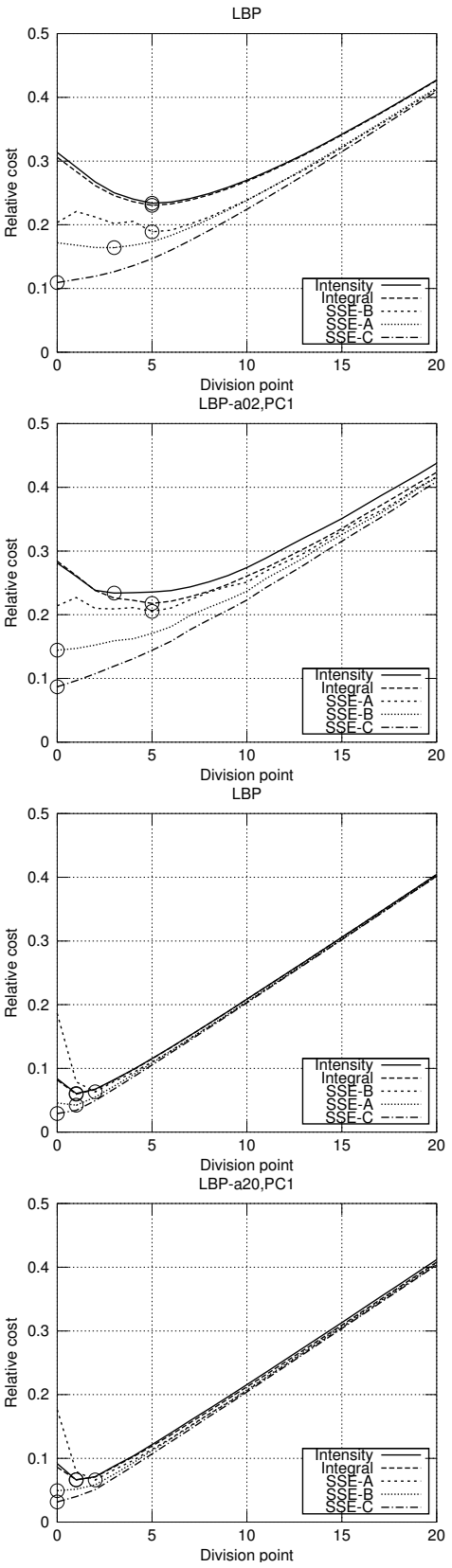

Figure 4: Comparison of optimization results with measurements on PC1. Each couple of plots shows (left) result of optimization and (right) result of measurement.

tection in a hardware unit, combination of this unit with software implementation in an embedded platform and fine tuning of this combination by using the cost evaluation and minimization. The possible criteria for the minimization include power consumption of the whole system and processing speed.

References

[1] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[2] L. Bourdev and J. Brandt. Robust object detection via soft cascade. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 236–243, 2005.

[3] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon. *Parallel programming in OpenMP*. Morgan Kaufmann Publishers, 1. edition, 2001.

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conferenceon Computer Vision and Pattern Recognition*, volume 1, pages 886–893, Washington, DC, USA, 2005. IEEE Computer Society.

[5] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, November 2000.

[6] J. Granát, A. Herout, M. Hradiš, and P. Zemčík. Hardware acceleration of adaboost classifier. In *Workshop on Multimodal Interaction and Related Machine Learning Algorithms (MLMI)*, pages 1–12, 2007.

[7] A. Herout, M. Hradiš, R. Juránek, and P. Zemčík. Implementation of the "local rank differences" image feature using simd instructions of cpu. In *Proceedings of Sixth Indian Conference on Computer Vision, Graphics and Image Processing*, page 9, 2008.

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

[9] A. Herout, R. Jošth, R. Juránek, J. Havel, M. Hradiš, and P. Zemčík. Real-time object detection on cuda. *Journal of Real-Time Image Processing*, 2011(3):159–170, 2011.

[10] A. Herout, R. Jošth, P. Zemčík, and M. Hradiš. Gp-gpu implementation of the "local rank differences" image feature. In *Proceedings of International Conference on Computer Vision and Graphics 2008*, Lecture Notes in Computer Science, pages 1–11. Springer Verlag, 2008.

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

[12] M. Hiromoto, K. Nakahara, H. Sugano, Y. Nakamura, and R. Miyamoto. A specialized processor suitable for adaboost-based detection with haar-like features. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.

[13] D. W. Hosmer and S. Lemeshow. *Applied logistic regression (Wiley Series in probability and statistics)*. Wiley-Interscience Publication, Sept. 2000.

[14] M. Hradiš. Framework for research on detection classifiers. In *Proceedings of Spring Conference on Computer Graphics*, pages 171–177. Comenius University in Bratislava, 2008.

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

[16] R. Juránek, A. Herout, and P. Zemčík. Impelementing local binary patterns with simd instructions of cpu. In *Proceedings of Winter Seminar on Computer Graphics*, page 5. West Bohemian University, 2010.

[17] R. Juránek, M. Hradiš, and P. Zemčík. *Real-Time Systems*, chapter Real-Time Object Detection with Classifiers, page 21. InTech Education and Publishing, 2012.

[18] F. Kadlček, R. Juránek, and P. Zemčík. Automatic synthesis of classifiers in fgpa. In *International Bata Conference for Ph.D. Students and Young Researchers*, page 12. Univerzita Tomáše Bati ve Zlíně, Zlín, CZ, 2011.

[19] C. P. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*, page 555, Washington, DC, USA, 1998. IEEE Computer Society.

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

[21] J. Reinders. *Intel threading building blocks : outfitting C++ for multi-core processor parallelism*. O'Reilly, 2007.

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

[23] J. Sochman and J. Matas. Waldboost - learning for time constrained sequential detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 150–156, Washington, DC, USA, 2005. IEEE Computer Society.

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

[25] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1:51–518 vol.1, 2001.

[26] A. Wald. *Sequential Analysis*. John Wiley and Sons, Inc., 1947.

[27] R. Xiao, L. Zhu, and H.-J. Zhang. Boosting chain learning for object detection. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 709, Washington, DC, USA, 2003. IEEE Computer Society.

[28] P. Zemčík, M. Hradiš, and A. Herout. Exploiting neighbors for faster scanning window detection in images. In *ACIVS 2010*, LNCS 6475, page 12. Springer Verlag, 2010.

[29] P. Zemčík and M. Žádník. Adaboost engine. In *Proceedings of FPL 2007*, page 5. IEEE Computer Society, 2007.

[30] L. Zhang, R. Chu, S. Xiang, S. Liao, and S. Z. Li. Face detection based on multi-block lbp representation. In *ICB*, pages 11–18, 2007.

Selected Papers by the Author

Polok Lukáš, Herout Adam, Zemčík Pavel, Hradiš Michal, Juránek Roman, Jošth Radovan. "Local Rank Differences" Image Feature Implemented on GPU. *In: Proceedings of the 10th International Conference on Advanced Concepts for Intelligent Vision Systems*, Berlin, Heidelberg, DE, Springer, 2008, p. 170-181, ISBN 978-3-540-88457-6

Herout Adam, Zemčík Pavel, Juránek Roman, Hradiš Michal. Implementation of the "Local Rank Differences" Image Feature Using SIMD Instructions of CPU. *In: Proceedings of Sixth Indian Conference on Computer Vision, Graphics and Image Processing*, Bhubaneswar, IN, IEEE CS, 2008, p. 9, ISBN 978-0-7695-3476-3

Herout Adam, Juránek Roman, Zemčík Pavel. Implementing the Local Binary Patterns with SIMD Instructions of CPU. *In: Proceedings of WSCG 2010*, Plzeň, CZ, ZČU v Plzni, 2010, p. 39-42, ISBN 978-80-86943-86-2

Herout Adam, Zemčík Pavel, Hradiš Michal, Juránek Roman, Havel Jiří, Jošth Radovan, Žádník Martin. Pattern Recognition, Recent Advances. Vienna, AT, IN-TECH, 2010, p. 111-136, ISBN 978-953-7619-90-9

Herout Adam, Jošth Radovan, Juránek Roman, Havel Jiří, Hradiš Michal, Zemčík Pavel. Real-time object detection on CUDA *In: Journal of Real-Time Image Processing*, Vol. 2011, No. 3, DE, p. 159-170, ISSN 1861-8200

Juránek Roman, Hradiš Michal, Zemčík Pavel. Real-Time Systems, Real-Time Algorithms of Object Detection using Classifiers Vienna, AT, IN-TECH, 2012, p. 111-136, ISBN 978-953-7619-90-9