EURASIP Journal on Applied Signal Processing 2005:7, 1035–1046

Automatic Hardware Implementation Tool for a Discrete Adaboost-Based Decision Algorithm

J. Mitéran
Le2i (UMR CNRS 5158), Aile des Sciences de l'Ingénieur, Université de Bourgogne, BP 47870, 21078 Dijon Cedex, France
Email: miteranj@u-bourgogne.fr

J. Matas
Center for Machine Perception—CVUT, Karlovo Namesti 13, Prague, Czech Republic
Email: matas@cmp.felk.cvut.cz

E. Bourennane
Le2i (UMR CNRS 5158), Aile des Sciences de l'Ingénieur, Université de Bourgogne, BP 47870, 21078 Dijon Cedex, France
Email: ebourenn@u-bourgogne.fr

M. Paindavoine
Le2i (UMR CNRS 5158), Aile des Sciences de l'Ingénieur, Université de Bourgogne, BP 47870, 21078 Dijon Cedex, France
Email: paindav@u-bourgogne.fr

J. Dubois
Le2i (UMR CNRS 5158), Aile des Sciences de l'Ingénieur, Université de Bourgogne, BP 47870, 21078 Dijon Cedex, France
Email: julien.dubois@u-bourgogne.fr

Received 15 September 2003; Revised 16 July 2004

We propose a method and a tool for automatic generation of hardware implementation of a decision rule based on the Adaboost algorithm. We review the principles of the classification method and we evaluate its hardware implementation cost in terms of FPGA's slice, using different weak classifiers based on the general concept of hyperrectangle. The main novelty of our approach is that the tool allows the user to find automatically an appropriate tradeoff between classification performances and hardware implementation cost, and that the generated architecture is optimized for each training process. We present results obtained using Gaussian distributions and examples from UCI databases. Finally, we present an example of industrial application of real-time textured image segmentation.

Keywords and phrases: Adaboost, FPGA, classification, hardware, image segmentation.

1. INTRODUCTION

In this paper, we propose a method of automatic generation of hardware implementation of a particular decision rule. This paper focuses mainly on high-speed decisions (approximately 15 to 20 nanoseconds per decision) which can be useful for high-resolution image segmentation (low-level decision function) or pattern recognition tasks in very large image databases. Our work—in grey in the Figure 1—is designed in order to be easily integrated in a system-on-chip, which can perform the full process: acquisition, feature extraction, and classification, in addition to other custom data processing.

Many implementations of particular classifiers have been proposed, mainly based on neural networks [1, 2, 3] or more recently on support vector machine (SVM) [4]. However, the implementation of a general classifier is not often optimum in terms of silicon area, because of the general structure of the selected algorithm, and a manual VHDL description is often a long and difficult task. During the last years, some high-level synthesis tools, which consist of translating a high-level behavioural language description into a

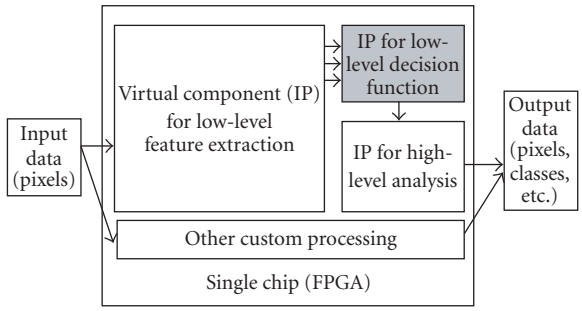


Figure 1: Principle of a decision function integrated in a system-on-chip.

register-transfer-level representation (RTL) [5], have been developed and which allow such a manual description to be avoided. Compilers are available for example for SystemC, Streams-C, Handel-C [6, 7], or for translation of DSP binaries [8]. Our approach is slightly different, since in the case of supervised learning, it is possible to compile the learning data in order to obtain the optimized architecture, without the need of a high-level language translation.

The aim of this work is to provide the EDA tool (Boost2VHDL, developed in C++) which generates automatically the hardware description of a given decision function, while finding an efficient tradeoff between decision speed, classification performances, and silicon area which we will call hardware implementation cost denoted as λ. The development flow is depicted in Figure 2. The idea is to generate automatically the architecture from the learning data and the results of the learning algorithm.

The first process is the learning step of a supervised classification method, which produces, off-line, a set of rules and constant values (built from a set of samples and their associated classes). The second step is also an off-line process. During this step, called Boost2VHDL, we built automatically from the previously processed rules the VHDL files implementing the decision function. In a third step, we use a standard implementation tool, producing the bit-stream file which can be downloaded in the hardware target. A new learning step will give us a new architecture. During the online process, the classification features and the decision function are continuously computed from the input data, producing the output class (see Figure 1).

This approach allows us to generate an optimized architecture for a given learning result, but implies the use of a programmable hardware target in order to keep flexibility. Moreover, the time constraints for the whole process (around 20 nanoseconds per acquisition/feature extraction/decision) imply a high use of parallelism. All the classification features have to be computed simultaneously, and the intrinsic operations of the decision function itself have to be computed in parallel. This naturally led us using FPGA as a potential hardware target.

In recent years, FPGAs have become increasingly important and have found their way into system design. FPGAs are used during development, prototyping, and initial production and can be replaced by hardwired gate arrays or application-specific component (ASIC) for high-volume production. This trend is enforced by rapid technological progress, which enables the commercial production of ever more complex devices [9]. The advantage of these components compared to ASIC is mainly their on-board reconfigurability, and compared to a standard processor, their high level of potential parallelism [10]. Using reconfigurable architecture, it is possible to integrate the constant values in the design of the decision function (here for example the constants resulting from the learning step), optimizing the number of cells used. We consider here the slice (Figure 3) as the main elementary structure of the FPGA and the unit of λ. One component can contain a few thousand of these blocks. While the size of these components is always increasing, it is still necessary to minimize the number of slices used by each function in the chip. This reduces the global cost of the system, increases the classification performance and the number of operators to be implemented, or allows the implementation of other processes on the same chip.

We choose the well known Adaboost algorithm as the implemented classifier. The decision step of this classifier consists in a simple summation of signed numbers [11, 12, 13]. Introduced by Schapire in 1990, Boosting is a general method of producing a very accurate prediction rule by combining rough and moderately inaccurate "rules of thumb." Most recent work has been on the "AdaBoost" boosting algorithm and its extensions. Adaboost is currently used for numerous researches and applications, such as the Viola-Jones face detector [14], or in order to solve the image retrieval problem [15] or the word-sense disambiguation problem [16], or for prediction in wireless telecommunications industry [17]. It can be used in order to improve classification performances of other classifiers such as SVM [18]. The reader will find a very large bibliography on http://www.boosting.org. Boosting, because of its interesting properties of maximizing margins between classes, is one of the most currently used and studied supervised method in the machine learning community, with support vector machine and neural networks. It is a powerful machine learning method that can be applied directly, without any modification, to generate a classifier implementable in hardware, and a complexity/performance tradeoff is natural in the framework: Adaboost learning constructs gradually a set of classifiers with increasing complexity and better performance (lower cross-validated error). All along this study, we kept in mind the necessity of obtaining high performances in terms of classification. We performed systematically measurements of classification error e (using a tenfold cross-validation protocol). Indeed, in order to follow real-time processing and cost constraints, we had to minimize the error e while minimizing the hardware implementation cost λ and maximize the decision speed. The maximum speed has been obtained using a fully parallel implementation.

In the first part of this paper, we present the principle of the proposed method, reviewing the Adaboost algorithm. We describe how it is possible, given the result of a learning step, to estimate the full parallel hardware implementation cost in terms of slices.

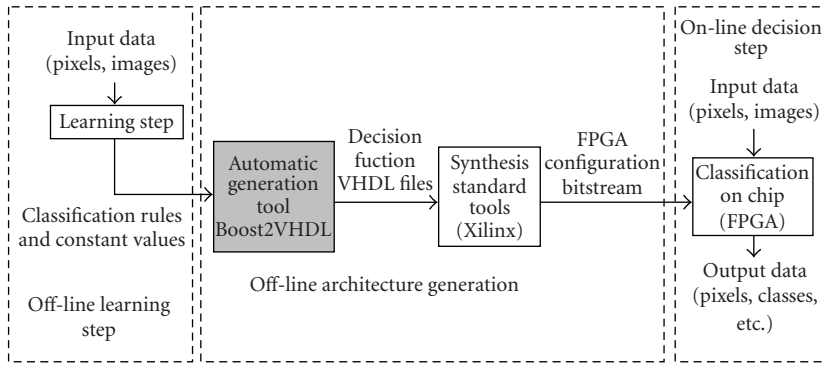


Figure 2: Development flow.

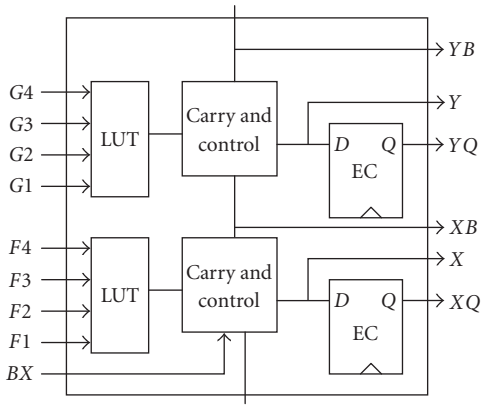


Figure 3: Slice structure.

In the second part, we define a family of weak classifiers suitable to hardware implementation, based on the general concept of hyperrectangle. We present the algorithm which is able to find a hyperrectangle which minimizes the classification error and allows us to find a good tradeoff between classification performance and the hardware implementation cost which we estimated. This method is based on a previous work: we have shown in [19, 20] that it is possible to implement a hyperrectangle-based classifier in a parallel component in order to obtain the required speed. Then, we define the global hardware implementation cost, taking into account the structure of the Adaboost method and the structure of the weak classifiers.

In the third part, results are presented: we applied the method on Gaussian distributions, which are often used in literature for performance evaluation of classifiers [21], and we presented results obtained on real databases coming from the UCI repository. Finally, we applied the method to an industrial problem, which consists in the real-time visual inspection of CRT cathodes. The aim is to perform a real-time image segmentation based on pixel classification. This segmentation is an important preprocessing used for detection of anomalies on the cathode.

The main contributions of this paper are the from-learning-data-to-architecture tool, and in the Adaboost process, the introduction of using hyperrectangles as a possible optimization of classification performances and hardware cost.

2. PROPOSED METHOD

2.1. Review of Adaboost

The basic idea introduced by Schapire and Freund [11, 12, 13] is that a combination of single rules or "weak classifiers" gives a "strong classifier." Each sample is defined by a feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_D)^T$ in a D-dimensional space and its corresponding class: $C(\mathbf{x}) = y \in \{-1, +1\}$ in the binary case.

We define the weighted learning set S of p samples as

$$S = \{(\mathbf{x}_1, y_1, w_1), (\mathbf{x}_2, y_2, w_2), \ldots, (\mathbf{x}_p, y_p, w_p)\}, \quad (1)$$

where w_i is the weight of the ith sample.

Each iteration of the process consists in finding the best possible weak classifier, that is, the classifier for which the error is minimum. If the weak classifier is a single threshold, all the thresholds are tested.

After each iteration, the weights of the misclassified samples are increased, and the weights of the well-classified sample are decreased.

The final class y is given by

$$y(\mathbf{x}) = \operatorname{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right), \quad (2)$$

where both α_t and h_t are to be learned by the boosting procedure presented in Algorithm 1. The characteristics of the classifier we have to encode in the architecture are the coefficients α_t for $t = 1, \ldots, T$, and the intrinsic constants of each weak classifier h_t.

2.2. Parallel implementation of the global structure

The final decision function to be implemented (equation (2)) is a particular sum of products, where each product is made of a constant (α_t) and the value -1 or $+1$ depending of the output of h_t. It is then possible to avoid computation of multiplications, which is an important gain in terms of hardware cost compared to other classifiers such as SVM or standard neural networks. The parallel structure of a possible hardware implementation is depicted in Figure 4.

(1) Input $S = \{(\mathbf{x}_1, y_1, w_1), (\mathbf{x}_2, y_2, w_2), \ldots, (\mathbf{x}_p, y_p, w_p)\}$, number of iteration T.
(2) Initialise $w_i^{(0)} = 1/p$ for all $i = 1, \ldots, p$.
(3) Do for $t = 1, \ldots, T$
(3.1) Train classifier with respect to the weighted samples set and obtain hypothesis

$$h_t : x \longrightarrow \{-1, +1\}.$$

(3.2) Calculate the weighted error ε_t of h_t:

$$\varepsilon_t = \sum_{i=1}^{p} w_i^{(t)} I(y_i \neq h_t(\mathbf{x}_i)).$$

(3.3) Compute the coefficient α_t:

$$\alpha_t = \frac{1}{2} \log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right).$$

(3.4) Update the weights

$$w_i^{(t+1)} = \frac{w_i^{(t)}}{Z_t} \exp\{-\alpha_t y_i h_t(\mathbf{x}_i)\},$$

where Z_t is a normalization constant:

$$Z_t = 2\sqrt{\varepsilon_t(1-\varepsilon_t)}.$$

(4) Stop if $\varepsilon_t = 0$ or $\varepsilon_t \geq 1/2$ and set $T = t - 1$.
(5) Output: $y(\mathbf{x}) = \operatorname{sgn}(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}))$.

Algorithm 1: The boosting procedure.

In terms of slices, the hardware cost can be expressed as follows:

$$\lambda = (T-1)\lambda_{\text{add}} + \lambda_T, \tag{3}$$

where λ_{add} is the cost of an adder (which will be considered as a constant here), and λ_T is the cost of the parallel implementation of the set of the weak classifiers:

$$\lambda_T = \sum_{t=1}^{T} \lambda_t, \tag{4}$$

where λ_t is the cost of the weak classifier h_t associated to the multiplexers. One can note that due to the binary nature of the output of h_t, it is possible to encode the results of additions and subtractions in the 16-bit LUT of FPGA, using the output of the weak classifiers as addresses (Figure 5). This is the first way to obtain an architecture optimized for a given learning result. The second way will be the implementation of the weak classifiers.

Since the classifier h_t is used T times, it is critical to optimize its implementation in order to minimize the hardware cost. As a simple classifier, single parallel-axis threshold is often used in the literature about Boosting. However, this type of classifier requires a large number of iterations T and hence the hardware cost increases (as it depends on the number of additions to be performed in parallel). To increase the complexity of the weak classifier allows faster convergence, and then minimizes the number of additions, but this will also increase the second member of the equation. We have then to find a tradeoff between the complexity of h_t and the hardware cost.

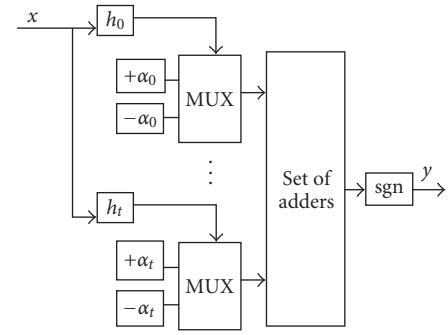


Figure 4: Parallel implementation of Adaboost.

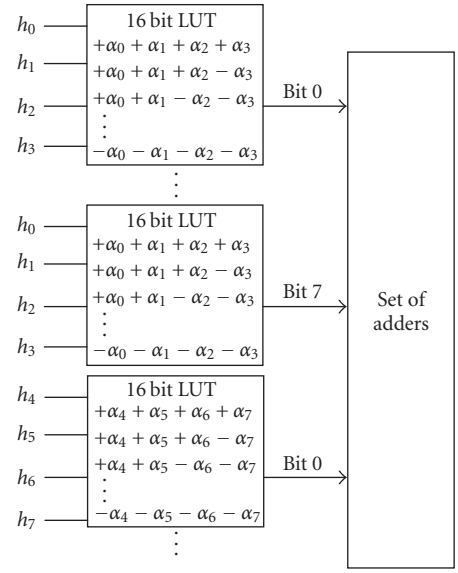


Figure 5: Details of the first stage: coding constants in the architecture of the FPGA.

3. WEAK CLASSIFIER DEFINITION AND IMPLEMENTATION OF THE WHOLE DECISION FUNCTION

3.1. Choice of the weak classifier: definitions

It has been proved in the literature that decision trees based on hyperrectangles (or union of boxes) instead of a single threshold give better results [22]. Moreover, the decision function associated with a hyperrectangle can be easily implemented in parallel (Figure 6).

However, there is no algorithm on the complexity of D which allows us to find the best hyperrectangle, that is, minimising the learning error. Therefore, we will use a suboptimum algorithm to find it.

We defined the generalized hyperrectangle as a set H of $2D$ thresholds and a class y_H, with $y_H \in \{-1, +1\}$:

$$H = \{\theta_1^l, \theta_1^u, \theta_2^l, \theta_2^u, \ldots, \theta_D^l, \theta_D^u, y_H\}, \tag{5}$$

where θ_k^l and θ_k^u are, respectively, the lower and upper limits of a given interval in the kth dimension. The decision

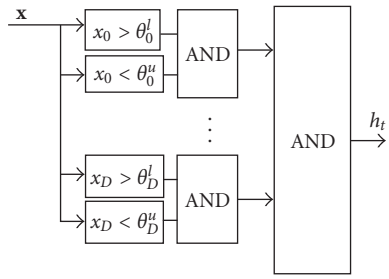


Figure 6: Parallel implementation of h_t.

function is

$$h_H(\mathbf{x}) = y_H \iff \prod_{d=1}^{D} ((x_d > \theta_d^l) \text{ and } (x_d < \theta_d^u)),$$
$$h_H(\mathbf{x}) = -y_H \quad \text{otherwise.} \tag{6}$$

This expression, where product is the logical operator, can be simplified if some of these limits are rejected to the infinite (or 0 and 255 in case of a byte-based implementation). Comparisons are not necessary in this case since the result will be always true. It is particularly important for minimizing the final number of used slices. Two particular cases of hyperrectangles have to be considered.

(i) The single threshold:

$$\Gamma = \{\theta_d, y_\Gamma\}, \tag{7}$$

where θ_d is a single threshold, $d \in \{1, \ldots, D\}$, and the decision function is

$$h_\Gamma(\mathbf{x}) = y_\Gamma \iff x_d < \theta_d,$$
$$h_\Gamma(\mathbf{x}) = -y_\Gamma \quad \text{otherwise.} \tag{8}$$

(ii) The single interval:

$$\Delta = \{\theta_d^l, \theta_d^u, y_\Delta\}, \tag{9}$$

where the decision function is

$$h_\Delta(\mathbf{x}) = y_\Delta \iff (x_d > \theta_d^l) \text{ and } (x_d < \theta_d^u),$$
$$h_\Delta(\mathbf{x}) = -y_\Delta \quad \text{otherwise.} \tag{10}$$

In these two particular cases, it is easy to find the optimum hyperrectangle, because each feature is considered independently from the others. The optimum is obtained by computing the weighted error for each possible hyperrectangle and choosing the one for which the error is minimum.

In the general case, one has to follow a particular heuristic given a suboptimum hyperrectangle. A family of such classifiers have been defined, based on the NGE algorithm described by Salzberg [23] whose performance was compared to the KNN method of Wettschereck and Dietterich [24]. This method divides the attribute space into a set of hyperrectangles based on samples. The performance of our own implementation was studied in [25]. We will review the principle of the hyperrectangle determination in the next section.

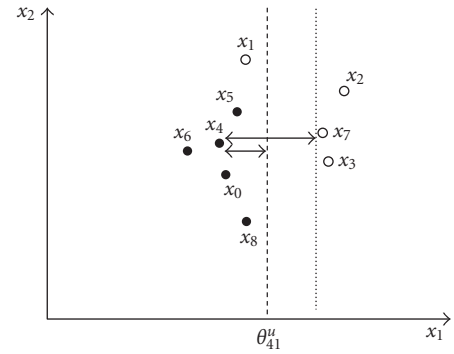


Figure 7: Determination of the first limit of $H(\mathbf{x}_4)$. In this case, $i = 4, z = 7, \tilde{k} = 1, \theta_{41}^u = R(x_{71} - x_{41})$.

3.2. Review of the hyperrectangle-based method

The core of the strategy is the hyperrectangles set S_H determination from a set of samples S.

The basic idea is to build around each sample $\{\mathbf{x}_i, y_i\} \in S$ a box or hyperrectangle $H(\mathbf{x}_i)$ containing no sample of opposite classes (see Figures 7 and 8):

$$H(\mathbf{x}_i) = \{\theta_{i1}^l, \theta_{i1}^u, \theta_{i2}^l, \theta_{i2}^u, \ldots, \theta_{iD}^l, \theta_{iD}^u, y_i\}. \tag{11}$$

The initial value is set to 0 for all lower bounds and 255 for all upper bounds.

In order to measure the distance between two samples in the feature space, we use the "max" distance defined by

$$d_\infty(\mathbf{x}_i, \mathbf{x}_j) = \max_{k=1,\ldots,D} |x_{ik} - x_{jk}|. \tag{12}$$

The use of this distance instead of the Euclidean distance allows building easily hyperrectangle instead of hypersphere. For all axes of the feature space, we determine the sample $\{\mathbf{x}_z, y_z\}$, $y_z \neq y_i$, as the nearest neighbour of $\mathbf{x}_i$ belonging to a different class:

$$z = \arg\min_j (d_\infty(\mathbf{x}_i, \mathbf{x}_j)). \tag{13}$$

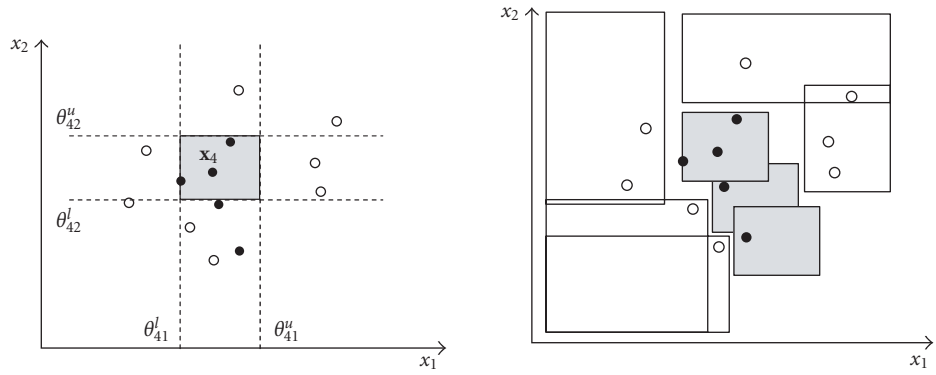
The threshold defining one bound of the box is perpendicular to the axis $\tilde{k}$ for which the distance is maximum:

$$\tilde{k} = \arg\max_k (|x_{ik} - x_{zk}|). \tag{14}$$

If $x_{i\tilde{k}} > x_{z\tilde{k}}$, we compute the lower limit $\theta_{i\tilde{k}}^l = R(x_{i\tilde{k}} - x_{z\tilde{k}})$. In the other case, we compute the upper limit $\theta_{i\tilde{k}}^u = R(x_{z\tilde{k}} - x_{i\tilde{k}})$.

The parameter R should be less than or equal to 0.5. This constraint ensures that the hyperrectangle cannot contain any sample of opposite classes.

The procedure is repeated until finding all the bounds of $H(\mathbf{x}_i)$.



(a) (b)

Figure 8: Hyperrectangle computation. (a) Determination of $H(\mathbf{x}_4)$. (b) Hyperrectangles obtained after merging step.

(3.1.1) Initialize $\varepsilon_{\min} = 1.0$
(3.1.2) Do for each class $y = -1, 1$
  Do for $i = 0, \ldots, q'(y)$
    Do for $j = i + 1, \ldots, q'(y)$
      Build $H_{\text{temp}} = H_i \cup H_j$
      Compute ε_H the weighed error based on H_{temp}
      if $\varepsilon_H < \varepsilon_{\min}$ then $H_{\text{opt}} = H_{\text{temp}}$ and $\varepsilon_H = \varepsilon_{\min}$
    end j
  end i
end y
(3.1.3) Output: $h_H = H_{\text{opt}}$

Algorithm 2

During the second step, hyperrectangles of a given class are merged together in order to eliminate redundancy (hyperrectangles which are inside of other hyperrectangles of the same class). We obtain a set S_H of hyperrectangles:

$$S_H = \{H_1, H_2, \ldots, H_q\}. \tag{15}$$

We evaluated the performance of this algorithm in various cases, using theoretical distributions as well as real sampling [19]. We compared the performance with neural networks, the KNN method, and a Parzen's kernel-based method [26]. It clearly appears that the algorithm performs poorly when the interclass distances are too small: an important number of hyperrectangles are created in the overlap area, slowing down the decision or increasing the implementation cost. However, it is possible to use the hyperrectangle generated as a step of the Adaboost process, selecting the best one in terms of classification error.

3.3. Boosting general hyperrectangle and combination of weak classifiers

From S_H we have to build one hyperrectangle H_{opt} minimizing the weighted error. To obtain this result, we merge hyperrectangles following a one-to-one strategy, thus building $q' = q(q-1)$ new hyperrectangles. We keep the hyperrectangle which gives the smallest weighted error.

For each iteration of the (3.1) Adaboost step, we design Algorithm 2.

(3) Do for $t = 1, \ldots, T$
  (3.1) Train classifier with respect to the weighted samples set $\{S, d^{(t)}\}$ and obtain the three hypothesis h_Γ, h_Δ, and h_H
  (3.2) Calculate weighted errors ε_Γ, ε_Δ, and ε_H introduced by each classifier
  (3.3) Choose h_t from $\{h_\Gamma, h_\Delta, h_H\}$ for which $\varepsilon_t = \min(\varepsilon_\Gamma, \varepsilon_\Delta, \varepsilon_H)$
  (3.4) Estimate λ

Algorithm 3

In order to optimize the final result, it is possible to combine the previous approaches, finding for each iteration the best weak classifier between the single threshold h_Γ, the interval h_Δ, and the general hyperrectangle h_H. Step (3) of the Adaboost algorithm is illustrated in Algorithm 3. As we will see in the results presented in Section 4, this strategy allows minimizing the number of iterations, and thus minimizing the final hardware cost in most of the case, even if the hardware cost of the implementation of an hyperrectangle is locally more important than the cost of the implementation of a single threshold.

3.4. Estimation of the hyperrectangle hardware implementation cost

As the elementary structure of the hyperrectangle is based on numerous comparisons performed in parallel (Figure 6), it is necessary to optimize the implementation of the comparator.

It is possible to estimate the hardware implementation cost of h_t taking into account that we can code the constant values of the decision function into the final architecture, using the advantage of FPGA-based reconfigurable computing. Indeed, the binary result L_B of the comparison of the variable byte A and the constant byte B is a function F_B of the bits of A:

$$L_B = F_B(A_7, A_6, \ldots, A_0). \tag{16}$$

We consider for example $B = 151$, 10010111 in binary, then, where "$*$" is the logic operator AND, "+" is the logic operator OR:

$$L_{151} = A_7{}^*(A_6 + (A_5 + (A_4{}^*A_3))), \tag{17}$$

L_{151} is true if A is greater than 151, and false otherwise.

More generally, we can write L_B as follows (for any byte B such that $0 < B < 255$):

$$L_B = A_7@(A_6@(A_5@(A_4@(A_3@(A_2@(A_1@(A_0@0))))))). \tag{18}$$

The @ operator denotes either the AND operator or the OR operator, depending on the position of @ and the value of B. In the worst case, the particular structure of L_B can be stored in two cascaded lookup tables (LUT) of 16 bits each (one slice).

We have coded in the tool Boost2VHDL a function which automatically generates a set of VHDL files: this is the hardware description of the decision functions h_t given the result of a training step (i.e., given the hyperrectangles limits). The files generated are used in the parallel architecture depicted in the Figure 5, which is also automatically generated using the constants of the Boosting process. We then have used a standard synthesizer tool for the final implementation in FPGA.

In the case of single threshold, $\lambda_t = 1$, for all $t \in [1, T]$. In the case of interval, $\lambda_t \leq 2$. In the case of general hyperrectangle, the decision rule requires in the worst case 2 comparators per hyperrectangle and per feature: $\lambda_t \leq 2D$.

3.5. Estimation of the global Adaboost implementation

Considering that some limits of the general hyperrectangle can be rejected to the "infinite," the general cost of the whole Adaboost-based decision can be expressed as follows:

$$\lambda \leq (T-1)\lambda_{\text{add}} + \mu T, \quad \text{with } \mu \leq 2D, \tag{19}$$

where μ is the sum of the number of lower limits of hyperrectangles which are greater than 0, and the number of upper limits which are lower than 255.

The implementation is efficient in terms of real-time computational for a reasonable value of D. In order to obtain very fast classification (around 10 nanoseconds per decision), we considered here only the full parallel implementation of all the process, including the classification features extraction (D features have to be computed in parallel). We limited our investigation here to $D = 64$.

One can note also that the hardware cost here is directly linked to the discrimination power of the classification features. In the classification framework, it is a well-known problem that is it critical to find efficient classification features in order to minimize classification error. Here, the better the classification features are selected, the faster the Boosting converges (T will be low), and the lower will be the hardware cost.

Moreover, an originality of this work is to allow the user to choose himself to control the Boosting process modifying the stopping criterion in step (4), and introducing a maximum hardware cost $\lambda_{\max}$. The step becomes

(4) Stop if $\varepsilon_t = 0$ or $\varepsilon_t \geq 1/2$ or $\lambda \geq \lambda_{\max}$ and set $T = t - 1$.

Finally, the user can choose the best tradeoff between classification error and hardware implementation cost for its application. Moreover, compared to a standard VHDL description of a classifier, our generated architecture is optimized for the user's application, since a specific VHDL description is generated for each process of training.

4. RESULTS

We applied our method in different cases. This first one is based on Gaussian distributions and in a two-dimensional space. We used this example in order to illustrate the method and the improvement given by hyperrectangle in terms of performance of classification.

The second series of examples, based on real databases coming from the UCI repository, is more significant in terms of hardware implementation, since they are performed in higher-dimensional spaces (until $D = 64$, this can be seen as a reasonable limit for a full parallel implementation).

The last example is from an industrial problem of quality control by artificial vision, where anomalies are to be detected in real time on metallic parts. The problem we focus on here is the segmentation step, which can be performed using pixelwise classification.

For each example, we also provide the result of a decision based on SVM developed by Vladimir Vapnik in 1979, which is known as one of the best classifiers, and which can be compared with Adaboost on the theoretical point of view. At the same time, SVM can achieve good performance when applied to real problems [27, 28, 29, 30]. In order to compare the implementation cost of the two methods, we evaluated the hardware implementation cost of SVM as

$$\lambda_{\text{SVM}} \simeq 72(3D-1)Ns + 8, \tag{20}$$

where Ns is the total number of "support vectors" determined during the training step. We used here an RBF-based kernel, using distance $L1$. While the decision function seems to be similar to the Adaboost one, the cost is here mainly higher because of multiplications; even if the exponential function can be stored in a particular lookup table to avoid computation, the kernel product K requires some multiplications and additions; the final decision function requires at least one multiplication and one addition per support vector:

$$C(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{Ns} y_i \alpha_i \cdot K(\mathbf{s}_i, \mathbf{x}) + b\right). \tag{21}$$

4.1. Experimental validation using Gaussian distributions

We illustrated the boosted hyperrectangle method using Gaussian distributions. The first tested configuration contains 4 classes in a two-dimensional feature space. An example of boundaries obtained using Adaboost and SVM is depicted in Figure 9. The second example is based on a classical XOR distribution, which is solved here using hyperrectangles.

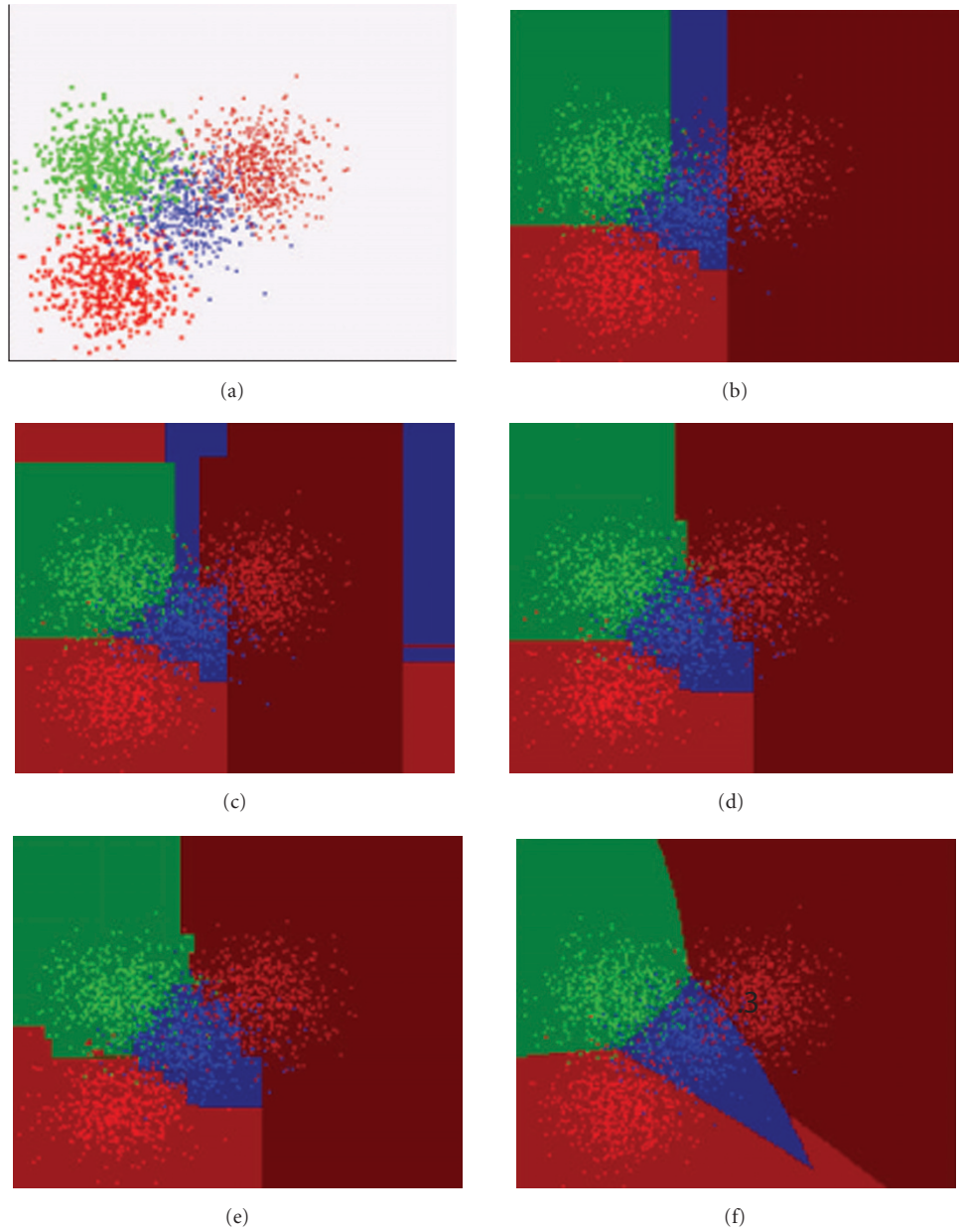


FIGURE 9: Example in $D = 2$, with 4 classes. (a) Original 4-class distribution, and boundaries with (b) single threshold, (c) single interval, (d) general hyperrectangle, (e) combination, and (f) SVM (RBF).

Results in terms of classification error are given in Table 1. As expected, the method works well in all the cases but the XOR one using single threshold or interval. We reported also the estimated number of slices, but in this particular case of a two-dimensional problem, it is clear that it is also possible to store the whole result of the SVM classifier in a single RAM, for example. However, this test well illustrates how it is possible to approximate complex classification boundaries with a single set of hyperrectangles.

4.2. *Experimental validation using real databases*

In order to validate our approach, we evaluate the hardware implementation cost of classification of databases from the UCI database repository. Results are summarized in the Table 2. We give the classification error e (%), the estimated number of slices (λ), a comparison with the decision time computation Pc, obtained with a standard PC (2.5 GHz) in the case of combination of best weak classifiers, and the speedup Su = Pc/0.01 of hardware computation, obtained with a 50 MHz clock.

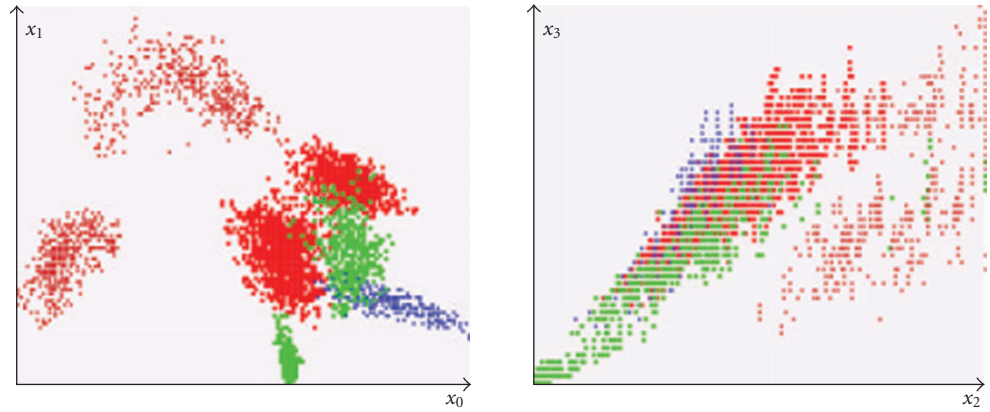
The dimension of the tested distributions is from 13 to 64, which seems to be a reasonable limit for byte-based full parallel implementation. The number of classes (C) is from 2 to 10. For each case, we give the result of classification using an RBF kernel-based SVM as a reference. One can see that the hardware cost of this classifier is not realistic here.

Table 1: Error using Gaussian distributions.

Distribution	D	Classes	Optimum	SVM (RBF)		Threshold		Interval		Hyperrectangle		Combination	
			e (%)	e (%)	λ_{SVM}	e (%)	λ	e (%)	λ	e (%)	λ	e (%)	λ
4 Gaussians	2	4	13	13.02	59048	14.8	181	13.62	386	13.22	46	13.2	32
Xor	2	2	4.4	4.6	129248	47.65	41	49	49	5.25	11	5.25	8

Table 2: Results on real databases.

Distribution	D	C	SVM (RBF)		Threshold		Interval		Hyperrectangle		Combination			
			e (%)	λ_{SVM}	e (%)	λ	e (%)	λ	e (%)	λ	e (%)	λ	Pc (μs)	Su
optdigit	64	10	1.15	20 215 448	2.605	5 292	2.735	5 414	2.59	4 392	2.255	4 379	873	43 650
pendigit	16	10	0.625	2 270 672	20.875	3 435	2.01	5 481	1.415	3 405	1.195	2 932	78	3 900
Ionosphere	34	2	7.95	465 416	8.23	126	6.81	149	7.095	119	5.68	88	1.13	56
IMAGE	17	7	3.02	1 699 208	12.91	568	7.655	697	4.015	973	5.085	778	4.0	200
WINE	13	3	4.44	87 560	3.33	98	5.525	98	6.11	18	3.325	36	1.5	75



(a) (b)

Figure 10: Extracted features for segmentation. (a) x_0 and x_1 projections. (b) x_2 and x_3 projections.

Considering the different results of our Adaboost implementation, it appears clearly that the combination of the three types of weak classifiers gives the better results. The optdigit and the pendigit cases can be solved using half of a circuit XCV600 of the VirtexE family, for example, while all the other cases can be implemented in a single low-cost chip.

Moreover, the classification error of the Adaboost-based classifier is very close to the SVM one.

Due to the parallel structure of our hardware implementation, the speedup is really important when the numbers of features D and classes C are high. Even if we reduce for example the frequency to 1 MHz in the case "optdigit" in order to follow a slower feature extraction, the speedup is still more than 800 compared to a standard software implementation.

Our system can also be used as a coprocessor embedded in a PCI-based board, limited to 33 MHz (32 bit data, allowing the parallel transmission of only 4 features from another board dedicated to data acquisition and features computation). The speedup in the case of image segmentation could be here for example:

$$\text{Su} = \frac{\text{Pc}/0.03}{D/4} = \frac{4/0.03}{17/4} \simeq 31. \tag{22}$$

However, the main interest of our method is to be integrated in a single component together with the other processes, as depicted in Figure 1.

4.3. Example of industrial application: image segmentation

We applied the previous method in order to perform an image segmentation step of a quality control process. The aim here is to detect some anomalies on manufactured parts, following the rate of 10 pieces per second. The resolution of the processed area is 300×300 pixels. The whole control (acquisition, feature extraction, segmentation, analysis, and final classification of the part) has to be achieved in less than 100 milliseconds. Thus, feature extraction and pixelwise classification have to be achieved in less than 1 microsecond.

In this application, "Good" texture and three types of anomalies of cathodes should be detected: bump ("Bump"), smooth surface ("Smooth"), and missing material ("Missing"). As detailed by Geveaux et al. in [26], the local mean of pixel luminance, the local mean of the Roberts gradient, and the local contrast, computed in a $[12 \times 12]$ neighborhood, have been selected to bring out the three types of anomalies. An example of projections of these features is presented on Figure 10.

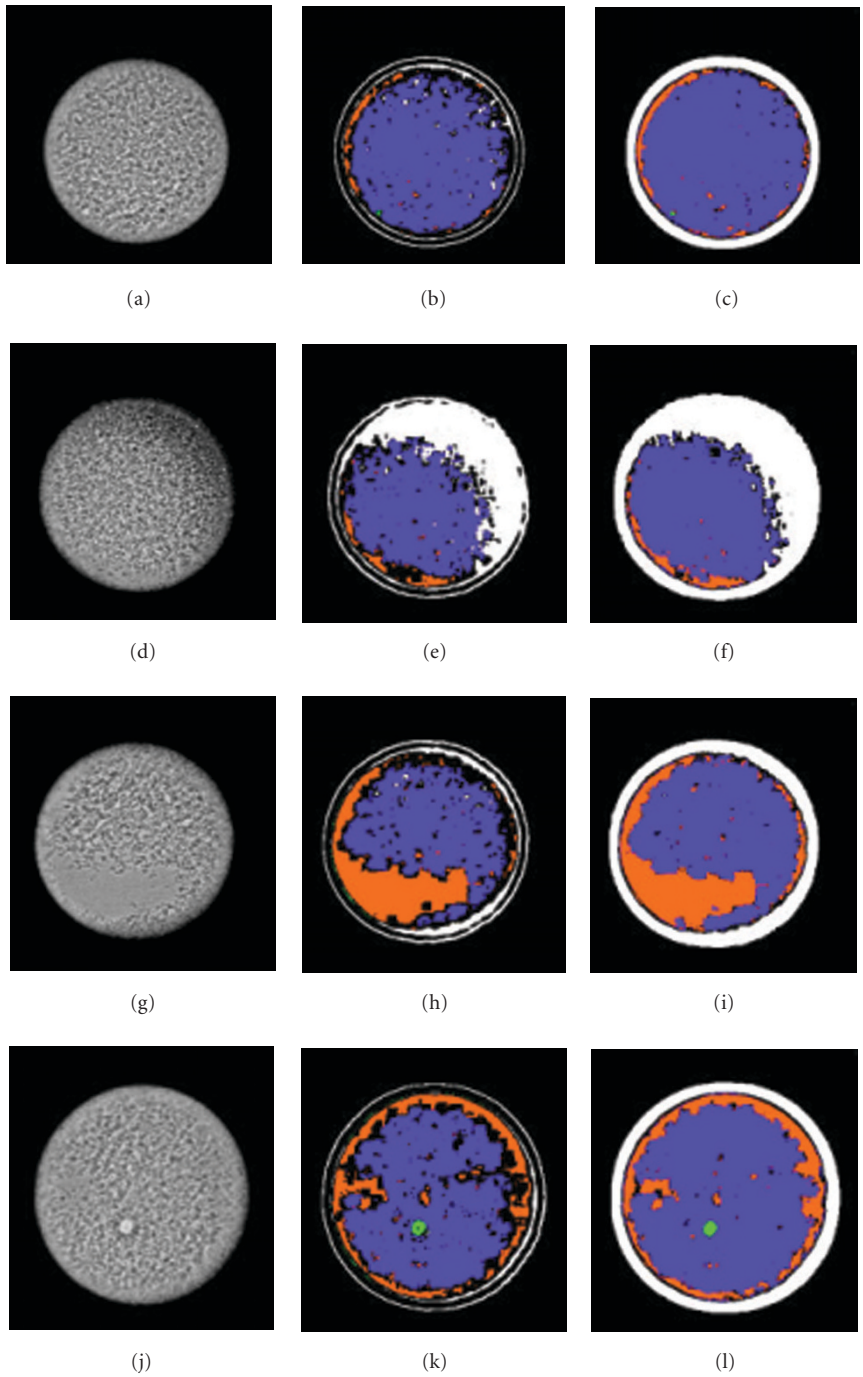


FIGURE 11: Example of segmentation results using threshold and hyperrectangles. (Left column) Original image with (a) denoting defect-free cathode, (d) missing material, (g) smooth area, and (j) bump. (Middle column) Image segmented using a single threshold. (Right column) Image segmented using hyperrectangles.

We depicted some examples of segmentation results in Figure 11. It is clear that the anomalies are better segmented using hyperrectangles than other weak classifiers. These results are confirmed by the cross-validated error presented in the Table 3. In this case, the better tradeoff between classification performance and hardware implementation cost is obtained using the combination of different weak classifiers. The estimated number of needed slices is less than 700 for a classification error $e = 2.44\%$, which is very close to the error obtained using SVM, and this for a very lower hardware cost than the SVM one.

One can see that the decision time of the standard PC implementation does not follow the real-time constraints (moreover, the features extraction time is not taken into

Table 3: Results on industrial application.

Distribution	D	Classes	SVM (RBF)		Threshold		Interval		Hyperrectangle		Combination			
			e (%)	λ	e (%)	λ	e (%)	λ	e (%)	λ	e (%)	λ	Pc (μs)	Su
Cathode	4	4	1.44	234440	8.16	434	6.15	467	2.41	726.5	2.44	677	2.7	135

account). The speedup of the hardware implementation—more than 100 for a 50 MHz clock—allows to follow these real-time constraints.

5. CONCLUSION

We have developed a method and EDA tool, called Boost2VHDL, allowing automatic generation of hardware implantation of a particular decision rule based on the Adaboost algorithm, which can be applied in many pattern recognition tasks, such as pixelwise image segmentation, character recognition, and so forth. Compared to a standard VHDL-based description of a classifier, the main novelty of our approach is that the tool allows the user to find automatically an appropriate tradeoff between classification performances and hardware implementation cost. Moreover, the generated architecture is optimized for the user's application, since a specific VHDL description is generated for each process of training.

We experimentally validated the method on theoretical distributions as well as real cases, coming from standard datasets and from an industrial application. The final error of this implemented classifier is close to the error obtained using an SVM-based classifier, which is often used in the literature as a good reference. Moreover, the method is really easy to use, since the only parameter to find is the choice of the weak classifier, the R value of the hyperrectangle-based method, or the maximum hardware cost allowed for the application. We are currently finalizing the development tool which will allow the development of the whole implementation process, from the learning set definition to FPGA-based implementation using automatic VHDL generation, and we will use it in the near future in order to speed up some processes using a coprocessing PCMCIA board based on a Virtex2 from Xilinx. Our future work will be the integration of this method as a standard IP generation tool for classification.

ACKNOWLEDGMENT

The author was supported by The Czech Academy of Sciences under project 1ET101210407.

REFERENCES

[1] P. Lysaght, J. Stockwood, J. Law, and D. Girma, "Artificial neural network implementation on a fine-grained FPGA," in *Proc. 4th International Workshop on Field-Programmable Logic and Applications (FPL '94)*, R. Hartenstein and M. Z. Servit, Eds., pp. 421–431, Prague, Czech Republic, September 1994.

[2] Y. Taright and M. Hubin, "FPGA implementation of a multilayer perceptron neural network using VHDL," in *Proc. 4th International Conference on Signal Processing (ICSP '98)*, vol. 2, pp. 1311–1314, Beijing, China, December 1998.

[3] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, UK, 1995.

[4] R. A. Reyna-Rojas, D. Dragomirescu, D. Houzet, and D. Esteve, "Implementation of the SVM generalization function on FPGA," in *Proc. International Signal Processing Conference (ISPC '03)*, pp. 147–153, Dallas, Tex, USA, March 2003.

[5] G. DeMichelli, *Synthesis and Optimization of Digital Circuits*, McGraw Hill, New York, NY, USA, 1994.

[6] J. Frigo, M. Gokhale, and D. Lavenier, "Evaluation of the stream-C C-to-FPGA compiler: an application perspective," in *Proc. 9th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGAs '01)*, pp. 134–140, Monterey, Calif, USA, February 2001.

[7] I. Page, "Constructing hardware-software systems from a single description," *Journal of VLSI Signal Processing*, vol. 12, no. 1, pp. 87–107, 1996.

[8] G. Mittal, D. C. Zaretsky, X. Tang, and P. Banerjee, "Automatic translation of software binaries onto FPGAs," in *Proc. 41st Design Automation Conference (DAC '04)*, pp. 389–394, San Diego, Calif, USA, June 2004.

[9] R. Enzler, T. Jeger, D. Cottet, and G. Tröster, "High-level area and performance estimation of hardware building blocks on FPGAs," in *Proc. 10th International Workshop on Field-Programmable Logic and Applications (FPL '00)*, vol. 1896 of *Lecture Notes in Computer Science*, pp. 525–534, Springer, Villach, Austria, August 2000.

[10] S. Hauck, "The roles of FPGAs in reprogrammable systems," *Proc. IEEE*, vol. 86, no. 4, pp. 615–638, 1998.

[11] R. E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, 1990.

[12] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[13] R. E. Schapire, "The boosting approach to machine learning: an overview," in *Proc. MSRI Workshop on Nonlinear Estimation and Classification*, pp. 149–172, Berkeley, Calif, USA, 2002.

[14] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '01)*, vol. 1, pp. 511–518, Kauai, Hawaii, USA, December 2001.

[15] K. Tieu and P. Viola, "Boosting image retrieval," *International Journal of Computer Vision*, vol. 56, no. 1–2, pp. 17–36, 2004.

[16] G. Escudero, L. Màrquez, and G. Rigau, "Boosting applied to word sense disambiguation, LNAI 1810," in *Proc. 12th European Conference on Machine Learning (ECML '00)*, pp. 129–141, Barcelona, Spain, 2000.

[17] M. C. Mozer, R. Wolniewicz, D. Grimes, E. Johnson, and H. Kaushansky, "Predicting subscriber dissatisfaction and improving retention in the wireless telecommunications industry," *IEEE Trans. Neural Networks*, vol. 11, no. 3, pp. 690–696, 2000.

[18] G. Rätsch, S. Mika, B. Schölkopf, and K.-R. Müller, "Constructing boosting algorithms from SVMs: an application to one-class classification," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 24, no. 9, pp. 1184–1199, 2002.

[19] J. Mitéran, P. Gorria, and M. Robert, "Classification géométrique par polytopes de contraintes intégration et performances," *Traitement du Signal*, vol. 11, no. 5, pp. 393–408, 1995.

[20] M. Robert, P. Gorria, J. Mitéran, and S. Turgis, "Architectures for a real time classification processor," in *Proc. IEEE Custom Integrated Circuits Conference (CICC '94)*, pp. 197–200, San Diego, Calif, USA, May 1994.

[21] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, NY, USA, 1973.

[22] I. De Macq and L. Simar, "Hyper-rectangular space partitionning trees, a few insight," discussion paper 1024, Université Catholique de Louvain, Belgium, 2002.

[23] S. Salzberg, "A nearest hyperrectangle learning method," *Machine Learning*, vol. 6, no. 3, pp. 251–276, 1991.

[24] D. Wettschereck and T. Dietterich, "An experimental comparison of the nearest-neighbor and nearest-hyperrectangle algorithms," *Machine Learning*, vol. 19, no. 1, pp. 5–27, 1995.

[25] J. Mitéran, J. P. Zimmer, F. Yang, and M. Paindavoine, "Access control: adaptation and real-time implantation of a face recognition method," *Optical Engineering*, vol. 40, no. 4, pp. 586–593, 2001.

[26] P. Geveaux, S. Kohler, J. Miteran, and F. Truchetet, "Analysis of compatibility between lighting devices and descriptive features using Parzen's Kernel: application to flaw inspection by artificial vision," *Optical Engineering*, vol. 39, no. 12, pp. 3165–3175, 2000.

[27] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, NY, USA, 1995.

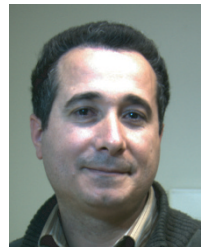
[28] B. Schölkopf, A. Smola, K.-R. Müller, C. J. C. Burges, and V. Vapnik, "Support vector methods in learning and feature extraction," *Australian Journal of Intelligent Information Processing Systems*, vol. 1, pp. 3–9, 1998.

[29] K. Jonsson, J. Kittler, Y. P. Li, and J. Matas, "Support vector machines for face authentication," in *Proc. British Machine Vision Conference (BMVC '99)*, T. Pridmore and D. Elliman, Eds., pp. 543–552, London, UK, September 1999.

[30] M. A. Hearst, B. Schölkopf, S. Dumais, E. Osuna, and J. Platt, "Trends and controversies - support vector machines," *IEEE Intell. Syst.*, vol. 13, no. 4, pp. 18–28, 1998.

J. Mitéran is an Associate Professor (HDR) at Laboratory Le2i, the University of Burgundy. He is involved in research about real-time implementation of pattern recognition algorithms, using reconfigurable computing. He is responsible for relationship between Le2i and industrial partners, and he is on the program committee of international conferences such as HSPP, QCAV, SPIE Machine Vision Applications in Industrial Inspection, and ECCV Workshop Applications of Computer Vision 2004.

J. Matas graduated (with honours) in technical cybernetics from the Czech Technical University in Prague, Czech Republic, in 1987, and received his Ph.D. degree from the University of Surrey, UK, in 1995. He has published more than 100 papers in refereed journals and conferences. He was awarded the science paper prize at the British Machine Vision Conference in 2002 and "The best scientific results of the Czech Technical University Prize" in 2003. He is on the program committee of a number of international conferences (ICPR, NIPS, CVPR, Face and Gesture Recognition, Audio- and Video-based Biometric Person Authentication). Dr. Matas was a Program Cochair for ECCV 2004—European Conference on Computer Vision.



E. Bourennane received the Ph.D. degree in automatics and image processing from the Le2i laboratory, the University of Burgundy in 1994. He is currently a Professor at the University of Burgundy. His research interests are mainly in real-time image processing. He is the President of the Program Committee of the AAA 2005 Workshop.

M. Paindavoine is a Professor at the University of Burgundy. He is teaching signal and image processing in the Engineering School ESIREM and IUP "Electronique et Image." He is the Head of the Laboratory Le2i (UMR CNRS 5158). His research interests are mainly in hardware implementation of signal and image processing using "adéquation algorithmes architectures" methodology. He is on the program committee of a number of international conferences (GRETSI, HSPP, QCAV, etc.)

J. Dubois received a Ph.D. degree from the University Jean Monnet of Saint Etienne. During his Ph.D., he developed a new image processing architecture named "Round-About" for real-time motion measurements. This architecture has been applied to measurement in fluid mechanics more precisely particle image velocimetry (PIV) in the University of Saint Etienne, France, in collaboration with the Image Processing and Expert System Laboratory (IPES), the University of Warwick, UK, where he has worked for six months. In March 2002, he joined EPFL based in Lausanne (CH) "Institut de Traitement des Signaux" (ITS) to develop a coprocessor, based on FPGA, for a new CMOS camera. Since 2003, he is an Associated Professor at Burgundy University. He is working on Codesign implementation and intelligent camera.