AUTOMATIC HARDWARE IMPLEMENTATION TOOL FOR A DISCRETE ADABOOST BASED DECISION ALGORITHM

J. Mitéran, J. Matas, E. Bourennane, M. Paindavoine, J. Dubois

Le2i - UMR CNRS 5158 Aile des Sciences de l'ingénieur
Université de Bourgogne - BP 47870 - 21078 Dijon - FRANCE
miteranj@u-bourgogne.fr
Center for Machine Perception – CVUT, Karlovo Namesti 13, Prague, Czech Republic

Abstract. 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 trade-off between classification performances and hardware implementation cost, and that the generated architecture is optimised 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 : 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 ns per decision) which can be useful for hi-resolution image segmentation (low level decision function) or pattern recognition tasks in very large image databases. Our work – in grey in the (Fig. 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.

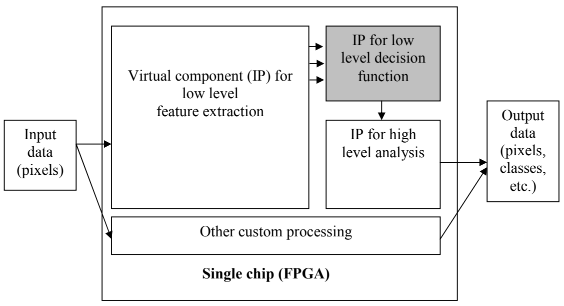

Input data (pixels)
Virtual component (IP) for low level feature extraction
IP for low level decision function
IP for high level analysis
Other custom processing
Single chip (FPGA)
Output data (pixels, classes, etc.)

Fig. 1 Principle of a decision function integrated in a System-On-Chip

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 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 trade-off between decision speed, classification performances and silicon area which we will call hardware implementation cost denoted as λ . The development flow is depicted in Fig. 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 on-line process, the classification features and the decision function are continuously computed from the input data, producing the output class (see Fig. 1).
This approach allows us to generate an optimised architecture for a given learning result, but imply the use of a programmable hardware target in order to keep flexibility. Moreover, the time constraints for the whole process (around 20 ns 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.

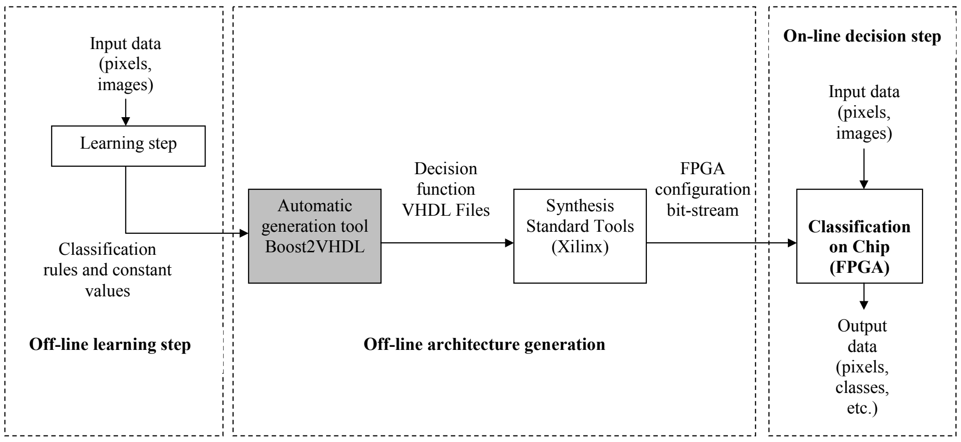

Input data (pixels, images)
Learning step
Classification rules and constant values
Off-line learning step
Automatic generation tool Boost2VHDL
Decision function VHDL Files
Synthesis Standard Tools (Xilinx)
FPGA configuration bit-stream
Off-line architecture generation
On-line decision step
Input data (pixels, images)
Classification on Chip (FPGA)
Output data (pixels, classes, etc.)

Fig. 2 Development flow

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), optimising the number of cells used. We consider here the slice (Fig. 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.

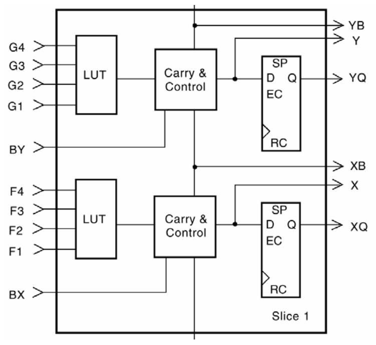


Fig. 3 Slice structure

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 Shapire 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 trade-off is natural in the framework: Adaboost learning constructs gradually a set of classifiers with increasing complexity and better performance (lower crossvalidated 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 ten-fold cross validation protocol). Indeed, in order to follow real-time processing and cost constraints, we had to minimise the error e while minimising the hardware implementation cost λ and maximise 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.

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 trade-off 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 hyperrectangles 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 present 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 optimisation 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 $\mathrm{x}=(x_1, x_2, \dots, x_\mathrm{D})^\mathrm{T}$ in an D dimensional space and its corresponding class:
$C(\mathrm{x}) = y \in \{-1, +1\}$ in the binary case.
We define the weighted learning set S of p samples as:

$$\mathrm{S} = \{(\mathrm{x}_1, y_1, w_1), (\mathrm{x}_2, y_2, w_2), \dots, (\mathrm{x}_p, y_p, w_p)\}.$$

Where w_i is the weight of the ith sample.
Each iteration of the process consists in finding the best weak classifier as possible, i.e. the classifier for which the error is minimum. If the weak classifier is a single threshold, all the thresholds are tested, and the
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:

$$y(\mathrm{x}) = sgn\left(\sum_{t=1}^{T} \alpha_t h_t(\mathrm{x})\right) \qquad (1)$$

Where both α_t and h_t are to be learned by the following boosting procedure.

```
1. Input $\mathrm{S} = \{(\mathrm{x}_1, y_1, w_1), (\mathrm{x}_2, y_2, w_2), \dots, (\mathrm{x}_p, y_p, w_p)\}$, number of iteration T
2. Initialise $w_i^{(0)} = \frac{1}{p}$ for all i=1, …, $p$
3. Do for t=1, …, T
    3.1 Train classifier with respect to the weighted samples set and obtain hypothesis
        $h_t : x \rightarrow \{-1, +1\}$
    3.2 Calculate the weighted error $\varepsilon_t$ of $h_t$:
        $\varepsilon_t = \sum_{i=1}^{p} w_i^{(t)} \mathrm{I}(y_i \neq h_t(\mathrm{x}_i))$
    3.3 Compute the coefficient $\alpha_t$
        $\alpha_t = \frac{1}{2} \log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
    3.4 Update the weights
        $w_i^{(t+1)} = \frac{w_i^{(t)}}{Z_t} \exp\{-\alpha_t y_i h_t(\mathrm{x}_i)\}$
        Where Zt is a normalization constant: $Z_t = 2\sqrt{\varepsilon_t(1-\varepsilon_t)}$
4. Stop if $\varepsilon_t = 0$ or $\varepsilon_t \geq \frac{1}{2}$ and set T=t-1
5. Output : $y(\mathrm{x}) = sgn\left(\sum_{t=1}^{T} \alpha_t h_t(\mathrm{x})\right)$
```

The characteristics of the classifier we have to encode in the architecture are the coefficients α_t for t=1, …, 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 (eq. 1) 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 Fig. 4.

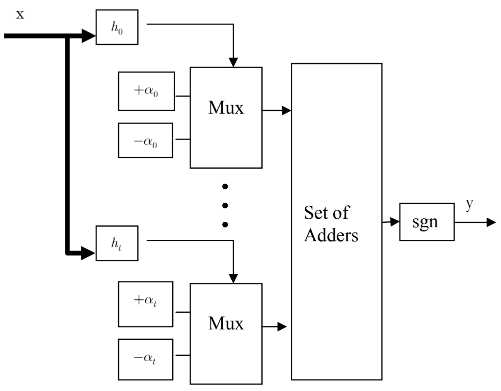


Fig. 4 Parallel implementation of Adaboost

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

$$\lambda = (T-1)\lambda_{add} + \lambda_T$$

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 :

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

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 (Fig. 5). This is the first way to obtain an architecture optimised for a given learning result. The second way will be the implementation of the weak classifiers.

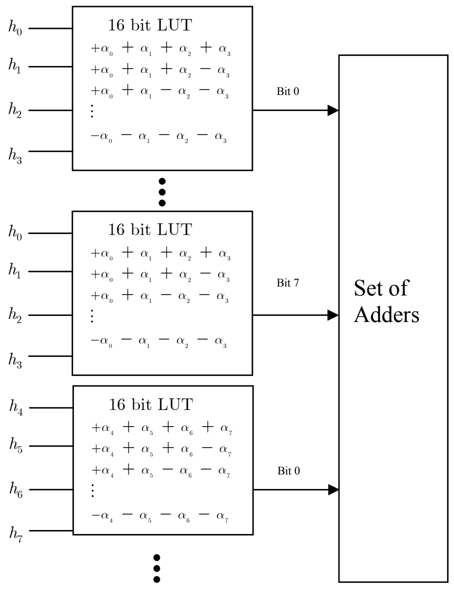

h_0
h_1
h_2
h_3
16 bit LUT
$+\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3$
$+\alpha_0 + \alpha_1 + \alpha_2 - \alpha_3$
$+\alpha_0 + \alpha_1 - \alpha_2 - \alpha_3$
$\vdots$
$-\alpha_0 - \alpha_1 - \alpha_2 - \alpha_3$
Bit 0
h_0
h_1
h_2
h_3
16 bit LUT
$+\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3$
$+\alpha_0 + \alpha_1 + \alpha_2 - \alpha_3$
$+\alpha_0 + \alpha_1 - \alpha_2 - \alpha_3$
$\vdots$
$-\alpha_0 - \alpha_1 - \alpha_2 - \alpha_3$
Bit 7
h_4
h_5
h_6
h_7
16 bit LUT
$+\alpha_4 + \alpha_5 + \alpha_6 + \alpha_7$
$+\alpha_4 + \alpha_5 + \alpha_6 - \alpha_7$
$+\alpha_4 + \alpha_5 - \alpha_6 - \alpha_7$
$\vdots$
$-\alpha_4 - \alpha_5 - \alpha_6 - \alpha_7$
Bit 0
Set of Adders

Fig. 5 Details of the first stage, coding constants in the architecture of the FPGA

Since the classifier h_t is used T time, it is critical to optimise its implementation in order to minimise 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 minimises the number of additions, but this will also increase the second member of the equation. We have then to find a trade off between the complexity of h_t and the hardware cost.

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 single threshold give better results [22]. Moreover, the decision function associated with a hyperrectangle can be easily implemented in parallel (Fig. 8).
However, there is no algorithm in the complexity of D which allows us to find the best hyperrectangle, i.e. minimising the learning error. Therefore we will use a suboptimum algorithm to find it.

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

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

Where θ_k^l and θ_k^u are respectively the lower and upper limits of a given interval in the *k*th dimension. The decision function is

$$h_H\langle x\rangle = y_H \Leftrightarrow \prod_{d=1}^{D}\left(\left(x_d > \theta_d^l\right)\text{and}\left(x_d < \theta_d^u\right)\right),\ h_H\langle x\rangle = -y_H \text{ otherwise}$$

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 minimising the final number of used slices. Two particular cases of hyperrectangles have to be considered:
The single threshold:

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

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

$$h_\Gamma\langle x\rangle = y_\Gamma \Leftrightarrow x_d < \theta_d,\ h_\Gamma\langle x\rangle = -y_\Gamma \text{ otherwise}$$

The single interval:

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

Where the decision function is:

$$h_\Delta\langle x\rangle = y_\Delta \Leftrightarrow \left(x_d > \theta_d^l\right)\text{and}\left(x_d < \theta_d^u\right),\ h_\Delta\langle x\rangle = -y_\Delta \text{ otherwise}$$

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 by 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 paragraph.

3.2 Review of Hyperrectangle based method

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

The basic idea is to build around each sample $\{x_i, y_i\} \in S$ a box or hyperrectangle $H(x_i)$ containing no sample of opposite classes (see Fig. 6 and Fig. 7):

$$H(x_i) = \{\theta_{i1}^l, \theta_{i1}^u, \theta_{i2}^l, \theta_{i2}^u, \ldots, \theta_{iD}^l, \theta_{iD}^u, y_i\}$$

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

$$d_\infty(x_i, x_j) = \max_{k=1,\ldots,D}|x_{ik} - x_{jk}|$$

The use of this distance instead of the Euclidian distance allows building easily hyperrectangle instead of hyper-sphere. For all axis of the feature space, we determine the sample $\{\mathrm{x}_z, y_z\}, y_z \neq y_i$ as the nearest neighbour of x_i belonging to a different class:

$$z = \arg\min_j \left(d_\infty \left(\mathrm{x}_i, \mathrm{x}_j \right) \right)$$

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

$$\tilde{k} = \arg\max_k \left(|x_{ik} - x_{zk}| \right)$$

if $x_{i\tilde{k}} > x_{z\tilde{k}}$ we compute the lower limit $\theta^l_{i\tilde{k}} = R.(x_{i\tilde{k}} - x_{z\tilde{k}})$. In the other case we compute the upper limit $\theta^u_{i\tilde{k}} = R.(x_{z\tilde{k}} - x_{i\tilde{k}})$.

The parameter R should be less 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 $H(\mathrm{x}_i)$.

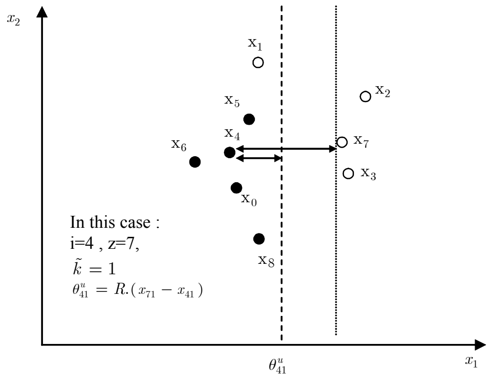


Fig. 6 Determination of the first limit of $H(\mathrm{x}_4)$

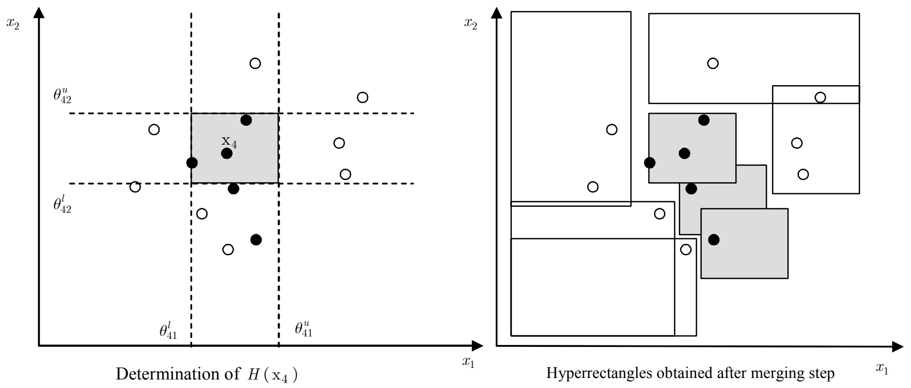


Fig. 7 Hyperrectangle computation

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

$$\mathrm{S_H} = \{H_1, H_2, ..., H_q\}$$

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 inter-class 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} minimising 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, the algorithm is:

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

In order to optimise 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. The step 3 of the Adaboost algorithm becomes:

```
3. Do for t=1, …, T
    3.1 Train classifier with respect to the weighted samples set $\{S, d^{(t)}\}$ and obtain the three hypothesis $h_\Gamma$, $h_\Delta$ and $h_H$
    3.2 Calculate weighted errors $\varepsilon_\Gamma$, $\varepsilon_\Delta$ and $\varepsilon_H$ introduced by each classifiers
    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 $\lambda$
```

As we will see in the results presented in the last paragraph, this strategy allows minimising the number of iterations, and thus minimising 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 (Fig. 8), it is necessary to optimise the implementation of the comparator.

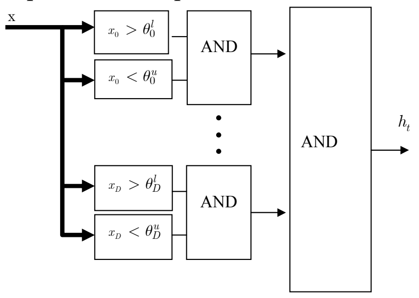


Fig. 8 Parallel implementation of h_t

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:

$L_B = F_B(A_7, A_6, \dots, A_0)$

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

$L_{151} = A_7 * (A_6 + (A_5 + (A_4 * A_3)))$

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 $0<B<255$):

$L_B = A_7 @ (A_6 @ (A_5 @ (A_4 @ (A_3 @ (A_2 @ (A_1 @ (A_0 @ 0)))))))$

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 Look Up 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 Fig. 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, $\lambda_t = 1, \forall t \in [1, T]$. 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 "infinit", the general cost of the whole Adaboost based decision can be expressed as follows:

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

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 reasonable value of D. Since in order to obtain very fast classification (around 10 ns per decision) we considered here only 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 is here 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 minimise 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 the step 4, and introducing a maximum hardware cost $\lambda_{\max}$. The step becomes:

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

Finally, the user can choose the best trade-off between classification error and hardware implementation cost for its application. Moreover, compared to a standard VHDL description of a classifier, our generated architecture is optimised 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 pixel-wised classification.

For each example, we also provide the result of a decision based on SVM developed by Vladimir Vapnik [REF] 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:

$$\lambda_{svm} \simeq 72(3D - 1)Ns + 8$$

Where *Ns* is the total number of "Support Vectors" determined during the training step. We used here a RBF based kernel, using distance L1. While the decision function seems to be similar to the Adaboost one's, the cost is here mainly higher because of multiplications, even if the exponential function can be stored in a particular look up 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:

$$\mathrm{C}\langle \mathrm{x} \rangle = \mathrm{Sgn}\left(\sum_{i=1}^{Ns} y_i \alpha_i \cdot K(\mathrm{s}_i, \mathrm{x}) + b \right)$$

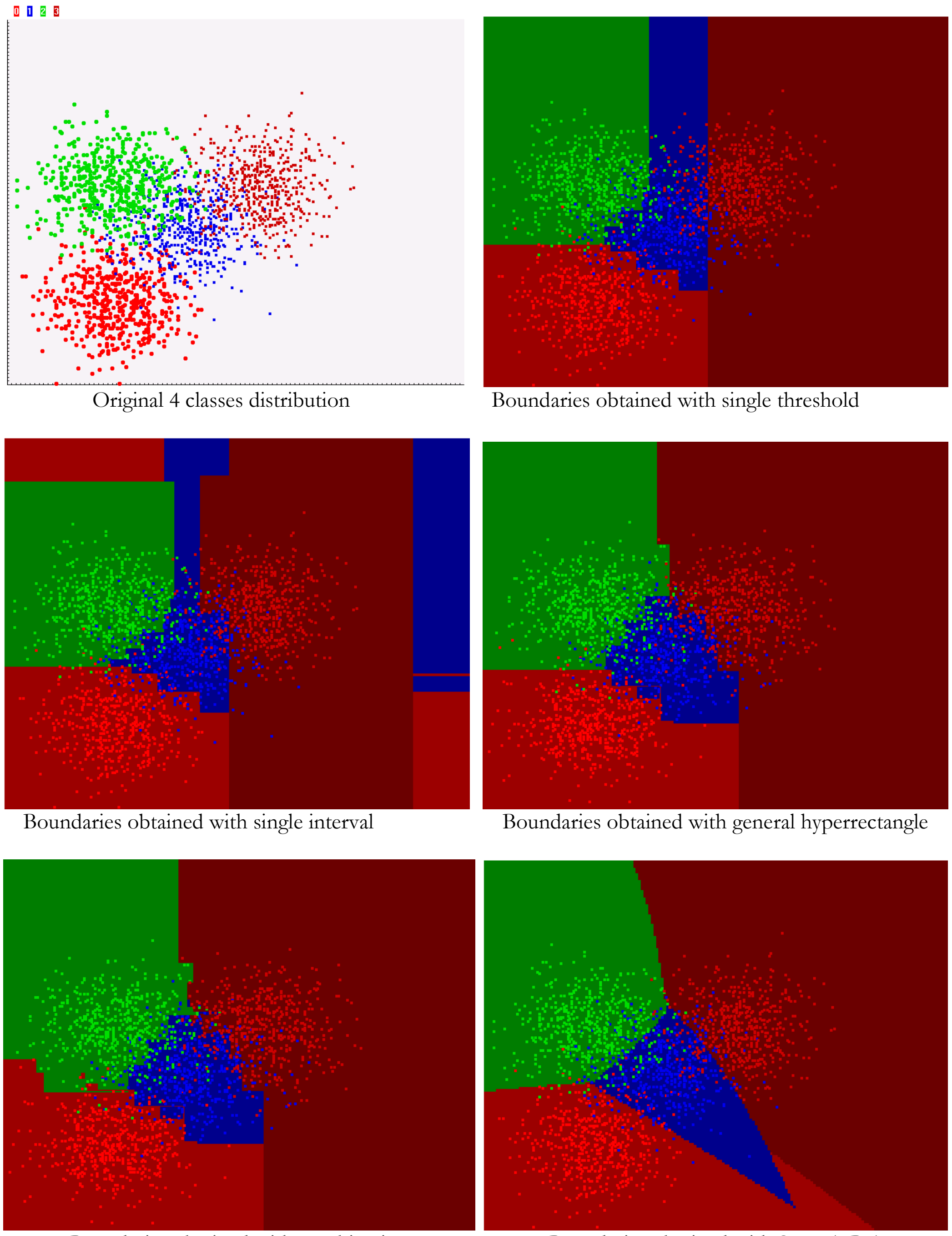
4.1 Experimental validation using Gaussian distributions

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 Fig. 9. The second example is based on a classical XOR distribution, which is solved here using hyperrectangles.

Table 1 Error using Gaussian distributions

	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

Results in term of classification error are given in the 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 single set of hyperrectangles.



Original 4 classes distribution

Boundaries obtained with single threshold

Boundaries obtained with single interval

Boundaries obtained with general hyperrectangle

Boundaries obtained with combination

Boundaries obtained with SVM (RBF)

Fig. 9 **Example in D=2, with 4 classes**

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 summarised 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 speed-up Su=Pc/0.01 of hardware computation, obtained with a 50MHz 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 a RBF kernel based SVM as a reference. One can see that the hardware cost of this classifier is not realistic here. 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 speed-up is really important when the numbers of features D and classes C are high. Even if we reduce for example the frequency to 1MHz for in the case "optdigit" in order to follow a slower feature extraction, the speed up is still more than 800 compared to 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 speed up in the case of image segmentation could be here for example:

$$Su = \frac{\frac{Pc}{0.03}}{\frac{D}{4}} = \frac{\frac{4}{0.03}}{\frac{17}{4}} \simeq 31$$

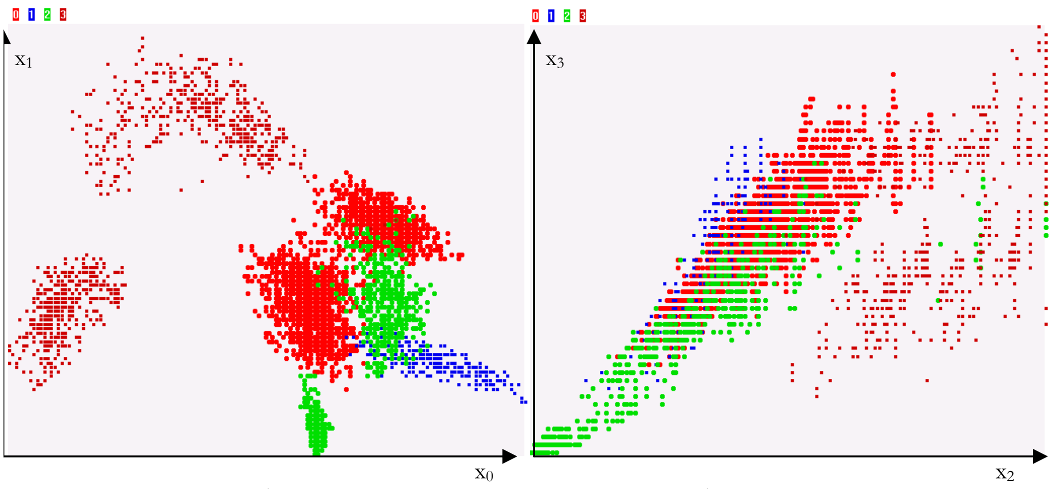
However, the main interest of our method is to be integrated in a single component together with the other processes, as depicted in fig. 1.

Table 2 Results on real databases

	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	20215448	2.605	5292	2.735	5414	2.59	4392	2.255	4379	873	43650
pendigit	16	10	0.625	2270672	20.875	3435	2.01	5481	1.415	3405	1.195	2932	78	3900
Ionosphere	34	2	7.95	465416	8.23	126	6.81	149	7.095	119	5.68	88	1.13	56
IMAGE	17	7	3.02	1699208	12.91	568	7.655	697	4.015	973	5.085	778	4.0	200
WINE	13	3	4.44	87560	3.33	98	5.525	98	6.11	18	3.325	36	1.5	75

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 300x300 pixels. The whole control (acquisition, feature extraction, segmentation, analysis and final classification of the part) has to be achieved in less than 100 ms. Thus, feature extraction and pixel wise classification have to be achieved in less than 1 µs.
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 in [19], the local mean of pixel luminance, the local mean of the Roberts gradient and the local contrast, computed in a [12 × 12] neighborhood, have been selected to bring out the three types of anomalies. An example of projections of these features is presented on Fig. 10.



x_0 and x_1 projections x_2 and x_3 projections

Fig. 10 Extracted features for segmentation

We depicted some examples of segmentation results in Fig. 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 trade-off between classification performance and hardware implementation cost is obtained using the combination of different weak classifiers. The estimated number of needed slice 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 that 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 account). The speed up of the hardware implementation - more than 100 for a 50 MHz clock - allows to follow these real-time constraints.

Table 3 Results on industrial application

	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

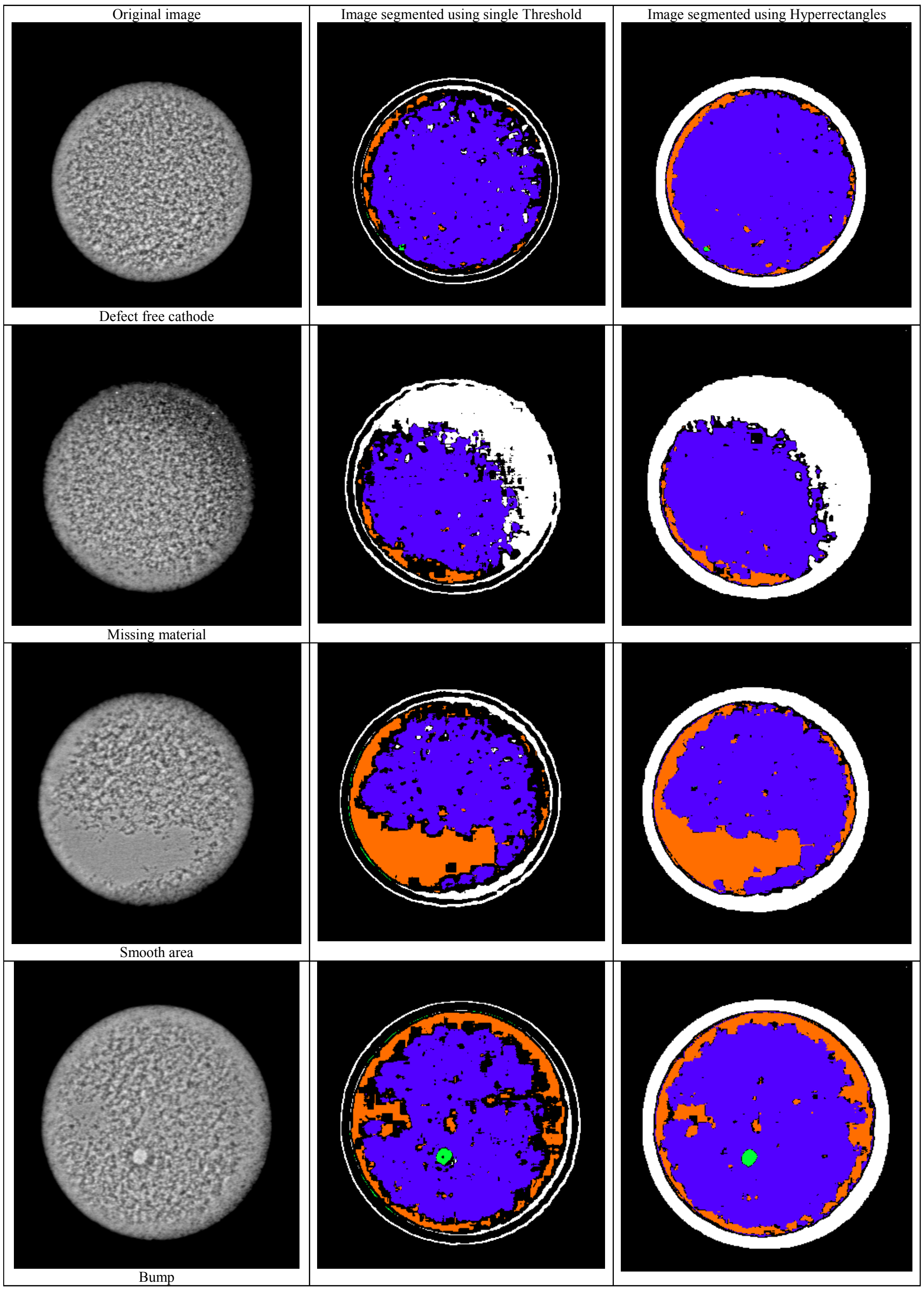

Original image
Image segmented using single Threshold
Image segmented using Hyperrectangles
Defect free cathode
Missing material
Smooth area
Bump

Fig. 11 Example of segmentation results using threshold and hyperrectangles

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 pixel wise image segmentation, character recognition, etc. 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 trade-off between classification performances and hardware implementation cost. Moreover, the generated architecture is optimised 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 a 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 parameters 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 finalising 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.

6 REFERENCES

[1] P. Lysaght, J. Stockwood, J. Law and D. Girma, Artificial Neural Network, Implementations on a Fine-Grained FPGA, in Field Programmable Logic and Applications, R. Hartenstein, M. Z. Servit (Eds.), Prague, pp. 421–431, 1994

[2] Y. Taright, M. Hubin, FPGA Implementation of a Multilayer Perceptron Neural Network using VHDL, 4th Int.l Conf. on Signal Processing (ICSP'98), Beijing,Vol 2, pp. 1311-1314, 1998.

[3] C. M. Bishop *Neural networks for Pattern Recognition*, Oxford University Press, 1995, pp. 110-230.

[4] R.A. Reyna-Rojas, D. Dragomirescu, D. Houzet, and D. Esteve, Implementation of the SVM generalization function on FPGA, International Signal Processing Conference (ISPC), Dallas (US), pp. 147-153, March 2003.

[5] G. DeMichelli, Synthesis and Optimization of Digital Circuits, MgGraw Hill, 1994.

[6] J. Frigo, M. Gokhale, D. Lavenier, Evaluation of the Stream-C C-to-FPGA compiler:An Application Perspective, Prof. 9th. Symp. On FPGAs, Monterey, CA, February 2001, pp. 134-140.

[7] I. Page, Constructing hardware-software systems from a single description, Journal of VLSI Signal Processing, 12(1), pp. 87-107, 1996.

[8] G. Mittal, D. C. Zaretsky, X. Tang, P. Banerjee, Automatic Translation of Software Binaries onto FPGAs, Design Automation Post Conference, San Diego (US), 2004.

[9] R. Enzler, T. Jeger, D. Cottet, and G. Tröster High-Level Area and Performance Estimation of Hardware Building Blocks on FPGAs, In *Field-Programmable Logic and Applications* (Proc. FPL 00), Lecture Notes in Computer Science, Vol. 1896, Springer, pp. 525-534, 2000.

[10] S. Hauck, The Roles of FPGAs in Reprogrammable Systems, Proceedings of the IEEE, 86(4): pp. 615-638, 1998.

[11] R.E. Schapire, The strengh of weak leanabilty Machine Learning, 5(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, 55(1), pp. 119-139, 1997.

[13] R.E. Schapire, The boosting approach to machine learning: An overview, MSRI Workshop on Nonlinear Estimation and Classification, 2002.

[14] P. Viola, M. Jones, Rapid object detection using a boosted cascade of simple features. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 511-518, December 2001.

[15] K. Tieu, P. Viola, Boosting Image Retrieval, International Journal of Computer Vision, 56(1-2), pp. 17-36, 2004.

[16] G. Escudero, L. Màrquez, and G. Rigau. Boosting applied to word sense disambiguation, LNAI 1810: Proceedings of the 12th European Conference on Machine Learning, ECML, Barcelona, Spain, pp. 129-141, 2000.

[17] M.C. Mozer, R. Wolniewicz, D. Grimes, E. Johnson, and H. Kaushanksy, Predicting subscriber dissatisfaction and improving retention in the wireless telecommunications industry, IEEE Transactions on Neural Networks, 11, 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 PAMI, 24(9), pp. 1184-1199, September 2002.
[19] J. Mitéran, P. Gorria and M. Robert, Classification géométrique par polytopes de contraintes. Performances et intégration , *Traitement du Signal*, Vol 11, pp. 393-408,1994.
[20] M. Robert, P. Gorria, J. Mitéran, S. Turgis Architectures for real-time classification processor, *Custom Integrated Circuit Conference*, San Diego CA, pp. 197-200, 1994.
[21] R. O. Duda and P.E. Hart, *Pattern classification and scene analysis*, Wiley, New York, 1973, pp. 230-243.
[22] I. De Macq, L. Simar, Hyper-rectangular space partitionning trees, a few insight, discussion paper 1024, Université Catholique de Louvain, 2002.
[23] S. Salzberg, A nearest hyperrectangle learning method. *Machine Learning*, 6: 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, n°1, pp. 5-27, 1995.
[25] J. Mitéran, J. P. Zimmer, F. Yang, M. Paindavoine Access control : adaptation and real-time implantation of a face recognition method, *Optical Engineering*, 40(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, n°12, pp. 3165-3175, 2000.
[27] V. Vapnik *The nature of statistical learning theory* , Springer-Verlag, New York, 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,* 1: pp. 3-9, 1998.
[29] K. Jonsson, J. Kittler, Y. P. Li, and J. Matas Support Vector Machines for Face Authentication. In T. Pridmore and D. Elliman, editors, British Machine Vision Conference, pp. 543-553, 1999.
[30] M. A. Hearst, B. Schölkopf, S. Dumais, E. Osuna and J. Platt, Trends and Controversies - Support Vector Machines. *IEEE Intelligent Systems*, 13(4) : pp. 18-28, 1998.

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