SVM Approximation for Real-time Image Segmentation by Using an Improved Hyperrectangles-based Method

J. Mitéran, S. Bouillant, E. Bourennane
Le2i - FRE CNRS 2309 Aile des Sciences de l'ingénieur
Université de Bourgogne
BP 47870
21078 Dijon - FRANCE
miteranj@u-bourgogne.fr

Summary

A real-time implementation of an approximation of the support vector machine decision rule is proposed. This method is based on an improvement of a supervised classification method using hyperrectangles, which is useful for real-time image segmentation. The final decision combines the accuracy of the SVM learning algorithm and the speed of a hyperrectangles-based method. We review the principles of the classification methods and we evaluate the hardware implementation cost of each method. We present the combination algorithm which consists of rejecting ambiguities in the learning set using SVM decision, before using the learning step of the hyperrectangles-based method. We present results obtained using Gaussian distribution and give an example of image segmentation from an industrial inspection problem. The results are evaluated regarding hardware cost as well as classification performances.

Running headline: SVM approximation for image segmentation

Introduction

Real-time image segmentation is a well known problem and can be solved using pixel-wise classification and specific classifiers. This paper focuses on very high speed decisions operators (approximately 10 ns per pixel) which, for example, can be used to detect anomalies on manufactured parts. The segmentation is usually the first step of a pattern recognition process.

Mainly because of the time constraint of our industrial application (10 images/s, size 1288x1080), only segmentation methods based on context-free local analysis of pixel neighbourhoods were considered. Classification is a central problem of pattern recognition [1] and many approaches to the problem have been proposed, e.g. neural networks [2], Support Vector Machines (SVM) [3], k-nearest neighbours (K-nn) and kernel-based methods. The chosen classifier must either be implemented in low-cost hardware or in optimised software running in real-time.

It has been shown that the SVM method gives very good results in many practical cases, [4], [5], [6], however, this robust algorithm is not often used for pixel-wise classification because of the decision rule complexity.

We previously developed a hyperrectangles-based classifier [7]: this hyperrectangle method belongs to the same family as the NGE (Nested Generalized Exemplars) algorithm, described by Salzberg [8], [9].

In [10], we showed that this classifier can be implemented as a parallel component in order to obtain the required speed, and in [11] we indicated that the performances are sufficient for use in a face recognition algorithm. However, the performance of the training step is sometimes affected by ambiguities in the training set, and, more generally, the basic hyperrectangles-based method is outperformed by the SVM algorithm.

We propose in this paper an original combination of classifiers allowing for fast and robust classification as applied to image segmentation. The SVM is used during a first step, pre-processing the training set and thus rejecting any ambiguities. The hyperrectangles-based learning algorithm is applied using the SVM classified training set. We will show that the hyperrectangle method imitates the SVM method in terms of performances, for a lower implementation cost using reconfigurable computing.

In the first part of this paper, we review the principles of the two classifiers: the Hyperrectangles-based method and the SVM. In the second part, we present our combination method as applied on Gaussian distributions, which are often used in literature for performance evaluation of classifiers [12] [1]. Finally, we present practical results obtained from the image segmentation of an industrial part.

Classification algorithms

Hyperrectangles-based method

This method divides the attribute space into a set of hyperrectangles for which simple comparators may easily satisfy the membership condition. This hyperrectangle method belongs to the same family as the NGE algorithm, described by Salzberg [9], whose performance was compared to the K-nn method by Wettschereck and Dietterich [8]. The performance of our own implementation was studied in [6].

The training step consists of collecting the set S of the most representative samples from the various classes and associating with each sample a local constraint (hyperrectangle) $H(\mathbf{x}_i)$.

$$S=\{(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),\dots,(\mathbf{x}_p,y_p)\}$$

Each sample is defined by a feature vector $\mathbf{x}$ in D dimensional space and its corresponding class or label $C(\mathbf{x})=y$:

$\mathbf{x}=(x_1, x_2, \dots, x_D)^T$

Hyperrectangle determination:

During the first step, a hyperrectangle $H(\mathbf{x})$ is built for each sample $\mathbf{x}$ as follows :

Each part Θ_p (see Figure 1) defines the area where

$d_\infty(\mathbf{x}_k,\mathbf{x}_l)=\left|x_p^k-x_p^l\right|$ with

$$d_\infty(x,y)=\max_{k=1,\dots,D}\left|x_k-y_k\right|$$

We define $\boldsymbol{\delta}_p$ as the nearest neighbour belonging to a different class in each quadrant Θ_p. If d_p is the distance between $\mathbf{x}$ and $\boldsymbol{\delta}_p$ in a given Θ_p, the limit of the hyperrectangle in the direction is computed as $d_f = d_p.R_p$.

The parameter R_p must be less than or equal to 0.5. This constraint ensures that the hyperrectangle cannot contain any samples of opposite classes.

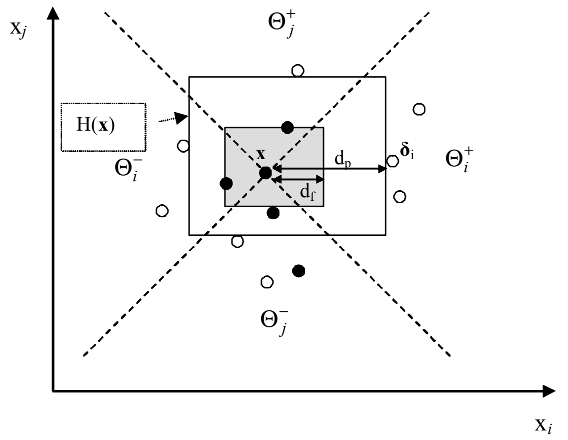


Figure 1: Hyperrectangle computation

During the second step, hyperrectangles of a given class are merged together in order to optimise the final number of hyperrectangles [7] (Figure 2).

The decision phase consists of allocating a class to a new attribute vector. The membership of a new feature value x_k to a given interval I_{ik} (ith hyperrectangle and kth feature) is easily verified by controlling the two following conditions : $(x_k > a_{ik})$ and $(x_k < b_{ik})$, where a_{ik} and b_{ik} are respectively the lower and upper limits of each polytope or hyperrectangle. Therefore, the verification of the membership of an unknown vector **x** to a class *y* results in a set of comparisons done simultaneously on each feature for every hyperrectangle of class *y*. The resulting decision rule is:

$$C(\mathbf{x}) = y \Leftrightarrow \sum_{i=1}^{i=m_y} \prod_{k=1}^{k=d} \left((x_k > a_{ik}).(x_k < b_{ik}) \right) \text{ is true} \quad (1)$$

where m_y equal to the number of hyperrectangles of class *y* after a merging phase. Sum and product are logical operators. This method is easy to use, and can be implemented for real-time classification using hardware [13] or software optimisation.

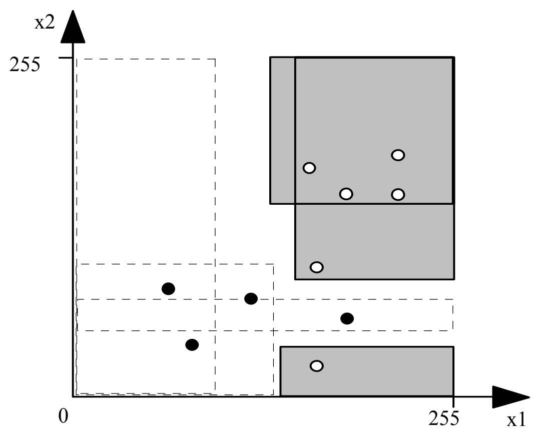


Figure 2: Hyperrectangles obtained in a two dimensional features space.

We developed an algorithm allowing evaluation of the implementation cost of this method in Field Programmable Gate Array (FPGA). 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 are replaced by hardwired gate arrays or application specific ICs (ASICs) for highvolume production [14]. The advantage of these components is mainly their reconfigurability [15].

It is possible to integrate the constant values (the limits of hyperrectangles) in the architecture of the decision function. We have coded a tool which automatically generates a VHDL description of a decision function given the result of a training step (i.e. given the hyperrectangles limits). We then used a standard synthesizer tool for the final implementation in FPGA. We verified that a single comparator between a variable (feature value) and a constant (hyperrectangle limit) only uses on average 0.5 slices (using bytes). The slice is the elementary structure of the FPGA of the Virtex family (Figure 3), and one component can contain a few thousand of these blocks. This estimation is possible since 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:

$L_B=F_B(A_7,A_6,...,A_0)$

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). On average we obtain 0.5 slices per comparison.

Since the decision rule requires 2 comparators per hyperrectangle and per dimension, we evaluate λ_H, the hardware cost of hyperrectangles implementation (number of slices) with:

$$\lambda_H = d\sum_{y=1}^{y=z} m_y \qquad (2)$$

where z is the number of classes. In the particular case of a 2-class problem, the summation can be computed only to y=z-1, since only one set of hyperrectangles defines the boundary.

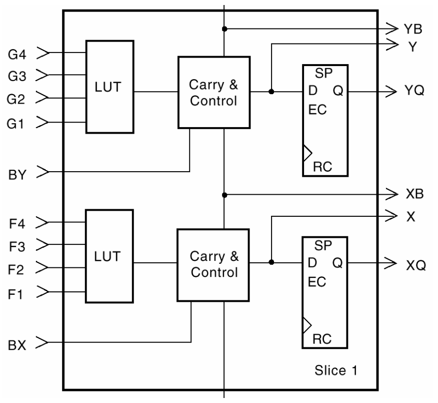
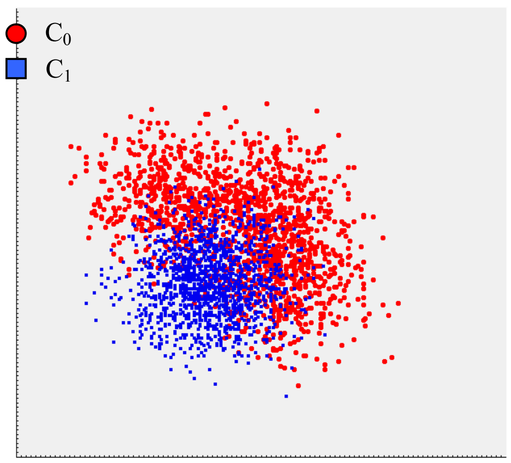


Figure 3: Slice structure

We evaluated the performance of this method in various cases, using theoretical distributions [6] as well as real sampling [11]. We compared the performance with neural networks, the K-nn method, and a Parzen's kernel based method [1]. It is clear that the algorithm performs poorly when the inter-class distances are too small. The overlap between classes is arbitrarily classified and introduces a classification error.



Multimodal Gaussian distributions

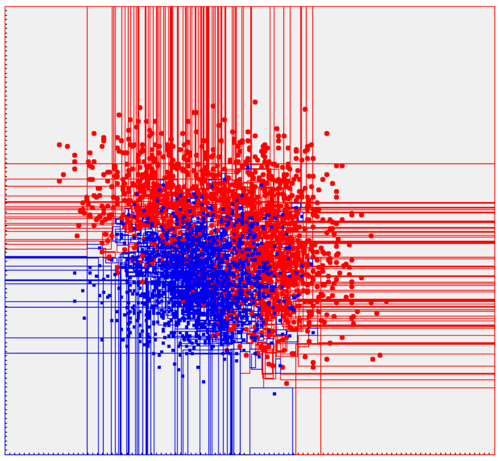

Hyperrectangle set

Figure 4: Gaussian Distributions

This is shown in Figure 4, where we present two distributions in a two-dimensional feature space. The C0 class is multimodal. The error is not symmetric, due to the priority given to the first class, and is rate is 18.88% for the C0 class, and 24.34% for the C1 class. Moreover, an important number of hyperrectangles are created in the overlap area, slowing down the decision or increasing the implementation cost.

Many classification methods, such as neural networks, density evaluation-based method, and the SVM described above are less sensitive to this overlap. It has been proven that a particular advantage of SVM over other learning algorithms is that it can be analyzed theoretically using concepts from computational learning theory and at the same time can achieve good performance when applied to real-world problems [16]. We chose this method as a pre-processing step and show that it is possible to approximate the result of the SVM using a combination of training steps.

SVM classification

A Support Vector Machine (SVM) is a universal learning machine developed by Vladimir Vapnik [3] in 1979. A review of the basic principles follows, considering a 2-class problem (whatever the number of classes, it can be reduced, by a "one-against-others" method, to a 2-class problem).

The SVM performs a mapping of the input vectors (objects) from the input space (initial feature space) R_d into a high dimensional feature space Q; the mapping is determined by a kernel function K. It finds a linear (or non-linear) decision rule in the feature space Q in the form of an optimal separating boundary, which leaves the widest margin between the decision boundary and the input vector mapped into Q. This boundary is found by solving the following constrained quadratic programming problem:

Maximize

$$\mathrm{W}(\alpha)=\sum_{i=1}^{n}\alpha_i-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i.\alpha_j.y_i.y_j.K(x_i,x_j) \quad (3)$$

Under the constraints

$$\sum_{i=1}^{n}\alpha_i.y_i=0$$

and $0 \le \alpha_i \le T$ for i=1, 2, …, n where $x_i \in R_d$ are the training sample set vectors, and $y_i \in \{-1,+1\}$ the corresponding class label. T is a constant needed for nonseparable classes. *K(u,v)* is an inner product in the feature space *Q* which may be defined as a kernel function in the input space. The condition required is that the kernel *K(u,v)* be a symmetric function which satisfies the following general positive constraint:

$$\iint_{R_d} K(\mathbf{u},\mathbf{v})\mathrm{g}(\mathbf{u})\mathrm{g}(\mathbf{v})\mathrm{d}\,\mathbf{u}\,\mathrm{d}\,\mathbf{v} > 0 \qquad (4)$$

which is valid for all $g \neq 0$ for which

$\int g^2(\mathbf{u})\ \mathrm{d}\mathbf{u} < \infty$ (Mercer's theorem).

The choice of the kernel $K(\boldsymbol{u},\ \boldsymbol{v})$ determines the structure of the feature space *Q*. A kernel that satisfies (3) may be presented in the form:

$$K(\mathbf{u},\mathbf{v}) = \sum_k a_k \Phi_k(\mathbf{u})\Phi_k(\mathbf{v}) \qquad (5)$$

where a_k are positive scalars and the functions Φ_k represent a basis in the space *Q*. Vapnik considered three types of SVMs [3]:

Polynomial SVM:

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}.\mathbf{y} + 1)^p \qquad (6)$$

Radial Basis Function SVM (RBF):

$$K(\mathbf{x},\mathbf{y}) = e^{\left(\frac{-\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}\right)} \qquad (7)$$

Two-layer neural network SVM:

$$K(\mathbf{x},\mathbf{y}) = Tanh\{k.(\mathbf{x}.\mathbf{y}) - \Theta\} \qquad (8)$$

The kernel should be chosen *a priori*. Other parameters of the decision rule (8) are determined by calculating (3), i.e. the set of numerical parameters $\{\alpha_i\}_1^n$ which determines the support vectors and the scalar *b*.

The separating plane is constructed from those input vectors, for which $\alpha_i \neq 0$. These vectors are called *support vectors* and reside on the boundary margin. The number *Ns* of support vectors determines the accuracy and the speed of the SVM. Mapping the separating plane back into the input space R_d, gives a separating surface which forms the following nonlinear decision rules:

$$\mathrm{C}(\mathbf{x}) = \mathrm{Sgn}\left(\sum_{i=1}^{Ns} y_i \alpha_i \cdot K(\mathbf{s}_i,\mathbf{x}) + b\right) \qquad (9)$$

where $\boldsymbol{s}_i$ belongs to the set of *Ns* support vectors defined in the training step.

One can see that the decision rule is easy to compute, but the cost of parallel implementation in ASIC or FPGA is clearly more important than in the case of the hyperrectangles based method. Even if the exponential function can be stored in a particular LUT in order to avoid computation, the scalar product K requires some multiplication and addition; the final decision function requires at least one multiplication and one addition per support vector. For a given model (set of support vectors), operators can be implemented using constant values (KCM [17]) as we did in the hyperrectangles-based method. However, the cost of multiplication is significantly more important than the comparator. Chapman, [3], [17], proposes a structure using 20 slices per 8 bit multiplier. An 8 bit adder uses 4 slices. The hardware cost of a possible SVM parallel implementation and total number of necessary slices is summarized in Table 1. We estimated the number of adders and multipliers needed by a fully parallel computation of K and the final sum of products, in the case of a simplified RBF kernel and a polynomial kernel. Given the number of slices needed by the computation of each elementary operator, we deduced λ_{svm}, the hardware cost of each implementation:

Table 1: SVM hardware cost estimation

		RBF (distance L1)	Polynomial degree p
		Number of operators	Number of operators
K (per support vector)	16-bit adders (8 slices)	-	d
	8-bit adders (4 slices)	3d-1	-
	Multiplier Kx8-bit (20 slices)	-	d
	Multiplier 8x8-bit (73 slices)	-	p-1
Sum of products	Multiplier Kx16-bit (72 slices)	Ns	Ns
	16 bits adders	1	1
		Number of slices	Number of slices
Total Slices		$\lambda_{svm} = 72(3d-1)Ns+8$	$\lambda_{svm} = (28d +73(p-1) +80)Ns+8$

Combination

Method

Combining decision classifiers is a classical way to increase performance of the general pattern recognition problem [18]. Three main methods are commonly used: sequential, parallel, and sequential-parallel combination (Figure 5). These approaches allow increased performance, but the cost of hardware implementation is high since all the decision functions have to be computed in order to obtain the final classification.

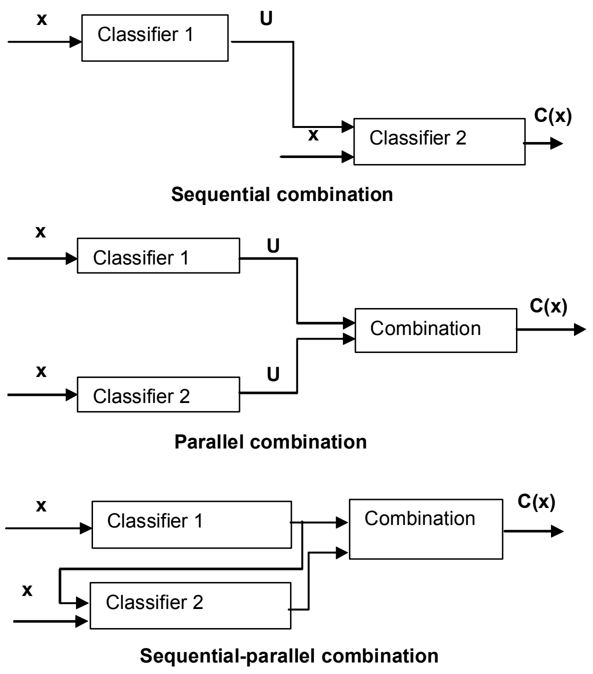


Figure 5: Combining decision

More generally, it is possible to combine classification methods during the training step [19]. We propose here such a combination, allowing an approximation of SVM decision boundaries using hyperrectangles (Figure 6).

Here SVM method is mainly used in order to reject ambiguities in the learning set. The algorithm combination is as follows:

- From a training set S $S=\{(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),\ldots,(\mathbf{x}_p,y_p)\}$, build a model M containing support vectors using SVM algorithm:

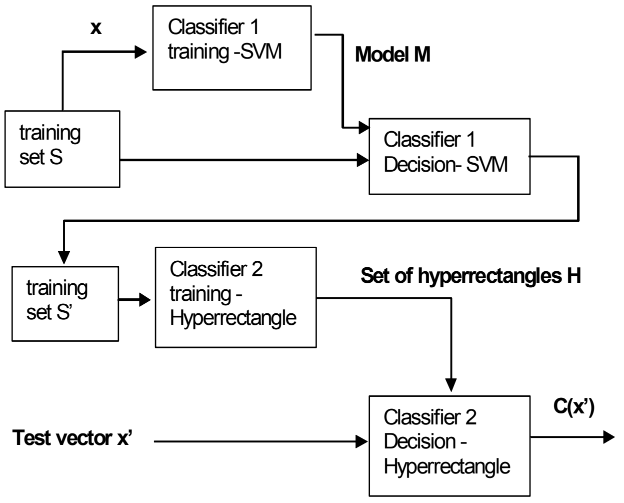
$$M=\{K,(\mathbf{s}_1,y_1),(\mathbf{s}_2,y_2),\ldots,(\mathbf{s}_{Ns},y_{Ns}),b\}$$

- build S', the new training set, classifying each sample of S using M and according to eq. (8). :

$$S' = \{(\mathbf{x}_1, y'_1), (\mathbf{x}_2, y'_2), \dots, (\mathbf{x}_p, y'_p)\},$$

- build H, set of hyperrectangles using S' and the algorithm described in paragraph 0.

During the decision phase, a new test vector **x** is classified regarding H and the decision rule (1).



x
Classifier 1 training -SVM
Model M
training set S
Classifier 1 Decision- SVM
training set S'
Classifier 2 training - Hyperrectangle
Set of hyperrectangles H
Test vector x'
Classifier 2 Decision - Hyperrectangle
C(x')

Figure 6: Combination of training steps

Application of Gaussian distributions

We validate the principle of the described method using Gaussian distributions. The first test configuration contains 2 classes, and the second configuration contains 3 classes. Results are summarized at the end of this section. In each case, we use cross-validation with p=1000 samples per class for the training set, and p=10000 samples per class for the test set.

Multimodal case

This learning set S is described in the previous paragraph. We use a RBF kernel (eq. (6)). The SVM classified set S' and the final set of hyperrectangles are depicted in Figure 7.

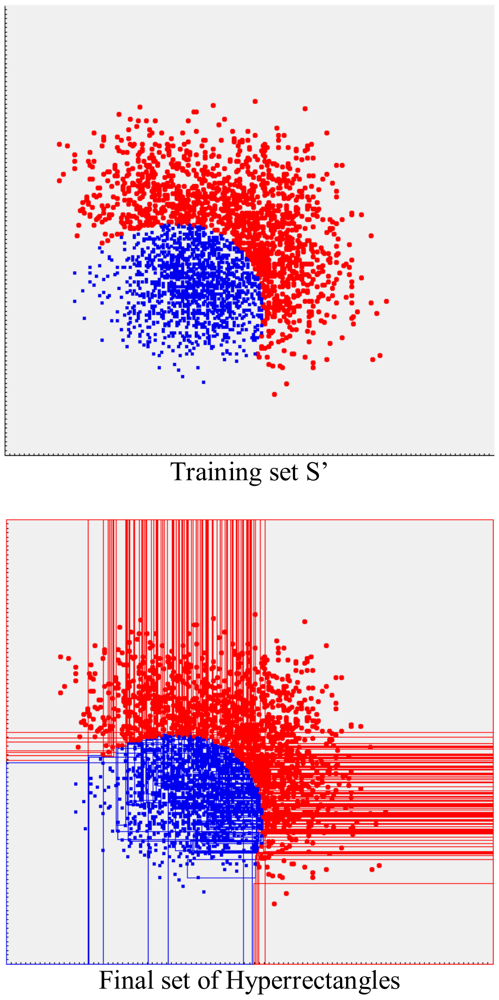


Figure 7: SVM and Hyperrectangle boundaries

The results show that the hyperrectangles-based method imitates the SVM decision algorithm, giving a good approximation of boundaries. The error rate of SVM is 17.27% for the C0 class and 13.74% for the C1 class while the errors obtained using final hyperrectangles (learning combination) are 17.16% and 13.96% respectively.

One can see that performances are very close, and more symmetric than when using the initial learning set.

Moreover, the number of hyperrectangles decreased, since the initial numbers were 748 (C0) and 762 (C1) before

SVM classification and only 104 (C0) and 101 (C1) after SVM classification. This allows the optimization of the hardware resources in the case of implementation. The cost of a direct implementation of SVM decision step is not comparable here, since the number of support vectors is 1190: the estimated hardware cost of SVM is λ_{svm}=428408 slices, whereas λ_H (hyperrectangles cost) equals only 205.

Case of 3 classes, monomodal distributions

The same method has been applied using three classes, in a two-dimensional space. The training set and the results of learning combination are represented in Figure 8.

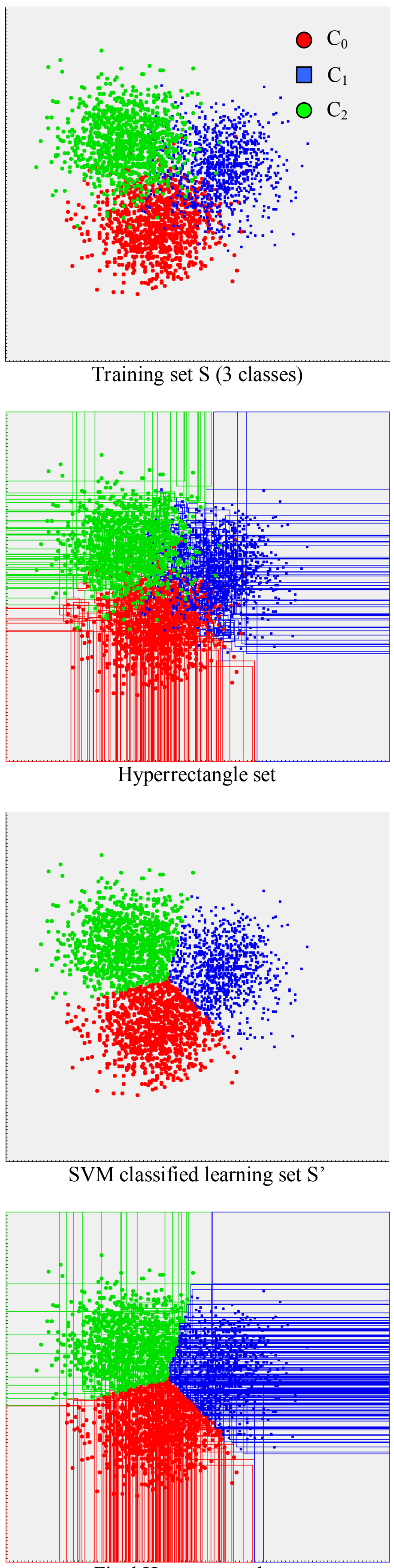


Figure 8 Training set S (3 classes)

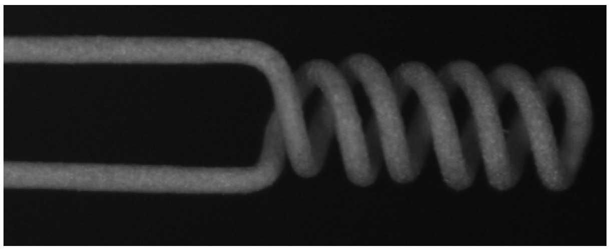
Error rates and numbers of hyperrectangles are reported in Table 2. The same improvement of performances is obtained in both cases, illustrating the good choice of the combination of training steps.

Table 2: Results using Gaussian distributions

		Direct learning using Hyperrectangle			Learning using SVM			Combination		
		c0	c1	c2	c0	c1	c2	c0	c1	c2
2 classes, D2	Error (%)	18.88	24.34	x	17.27	13.74	x	**17.16**	**13.96**	**x**
	nh	748	762	x	x	x	x	**104**	**101**	**x**
3 classes, D2	Error (%)	15.02	15.44	14.36	11.66	11.84	8.28	**11.60**	**11.70**	**8.62**
	nh	389	343	343	x	x	x	**92**	**86**	**71**

Real-time image segmentation for detection of anomalies

We applied our method to a preprocessing step of industrial project of quality control by artificial vision. The part we must control is a wire made up of a spiral part, *body* and a non-spiral part, *legs*.



Legs Body (spiraled part)

Figure 9: Part to be controlled.

All anomalies existing over the whole part can be grouped into 3 categories:

- Dimensional anomalies: diameter of the wire composing the part, length of the part, length of the *body*, length of the non-spiraled part.

- Visual anomalies discoloration, stripes, cracks, flaws of surface.

- Various anomalies of the spiral not comprising a deterioration of the wire composing the body.

We have to distinguish between thirty anomalies at the end of the project. During this preliminary work, we need to obtain a segmented image allowing extraction of high level classification features, such as distance between turns, surfaces and orientation of the turns, etc. Some of these features are depicted in Figure 10, where the part is modeled using whorls orientations and distances.

Note that the wire is textured: a single threshold could not be a robust operator. We extracted some simple texture features, keeping in mind real-time constraints.

The image size is 1288x1080, and the acquisition rate is 10 images/s.

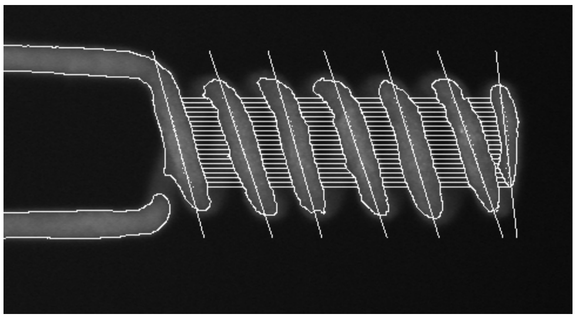


Figure 10: Part modeling

A preliminary study of segmentation features led us to choose a four-dimensional features space:

x_0 is the mean of luminance in 8x8 windows,

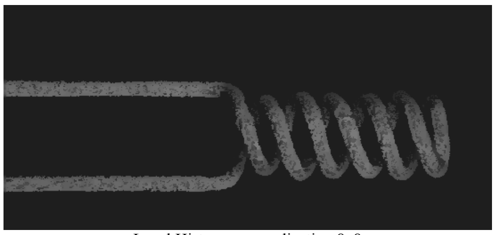
x_1 is the new pixel value after local histogram equalisation,

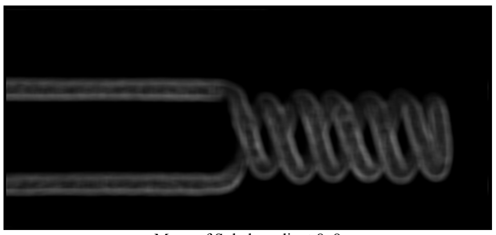
x_2 is the mean of a Sobel filter in 8x8 windows,

x_3 is the mean of the local contrast in 8x8 windows.

An example of these features is depicted in Figure 11.

Mean of luminance 8x8


Local Histogram equalization 8x8


Mean of Sobel gradient 8x8

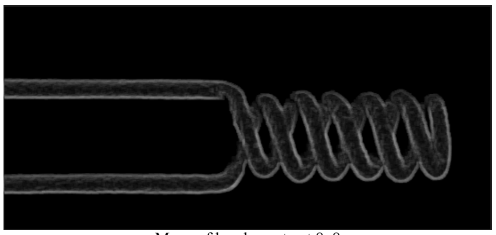

Mean of local constrast 8x8

Figure 11 Features used for segmentation

The local contrast *V(i,j)* in an [n x m] neighbourhood of *A(i,j)* pixel can be expressed as follows:

$$V(i,j) = \frac{A_{\max} - A_{\min}}{A_{\max} + A_{\min}} \tag{9}$$

with

$$A_{\max} = \max\left\{ \begin{array}{l} A(i+k, j+l), -\left\lfloor \frac{n-1}{2} \right\rfloor \le k \le \left\lfloor \frac{n}{2} \right\rfloor, \\ -\left\lfloor \frac{m-1}{2} \right\rfloor \le l \le \left\lfloor \frac{m}{2} \right\rfloor \end{array} \right\},$$

$$A_{\min} = \min\left\{ \begin{array}{l} A(i+k, j+l), -\left\lfloor \frac{n-1}{2} \right\rfloor \le k \le \left\lfloor \frac{n}{2} \right\rfloor, \\ -\left\lfloor \frac{m-1}{2} \right\rfloor \le l \le \left\lfloor \frac{m}{2} \right\rfloor \end{array} \right\} \tag{10}$$

and $\lfloor x \rfloor$ refers to the integer part of x (floor operator).

The local mean of the Sobel gradient norm *G(i,j)* and the local mean of luminance *S(i,j)* in a [n x m] neighbourhood of *A(i,j)* pixels can be written as follows:

$$S(i,j) = \frac{1}{mn} \sum_{k=p(n)}^{q(n)} \sum_{l=p(m)}^{q(m)} A(i+k, j+l) \tag{11}$$

and

$$G(i,j) = \frac{1}{mn} \sum_{k=p(n)}^{q(n)} \sum_{l=p(m)}^{q(m)} g(i+k, j+l) \tag{12}$$

with

$p(n) = -\left\lfloor \frac{n-1}{2} \right\rfloor$, $q(n) = \left\lfloor \frac{n}{2} \right\rfloor$, and *g(i,j)* is the Sobel gradient norm of the pixel *A(i,j)*.

We have chosen this set of features using the SFS, [20], [21], algorithm from a superset of 30 features (including variations of window size and other operators such as morphological operators, local entropy, etc).

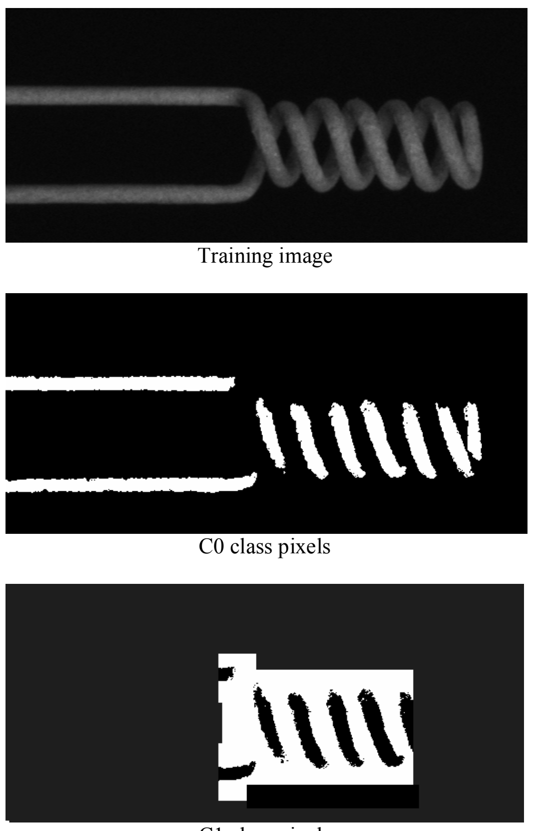


Figure 12: Training image and areas

We defined the labels of the training set S manually, using two binary images which define respectively class 0 and class 1 pixels (white pixels in Figure 12). For each class, 5000 pixels or samples were randomly chosen from the white areas of these pictures.

We have depicted two projections of the training set in Figure 13.

We applied our combination method, as described in the previous section, using 10 test images. An example test image is shown in Figure 14.

The SVM kernel used in this application is polynomial, degree 2.

The segmentation results are depicted in Figure 15. In order to quantify the results, we manually segmented the test images and computed the classification error of each class for the different segmented images. The obtained results are summarized in Table 3.

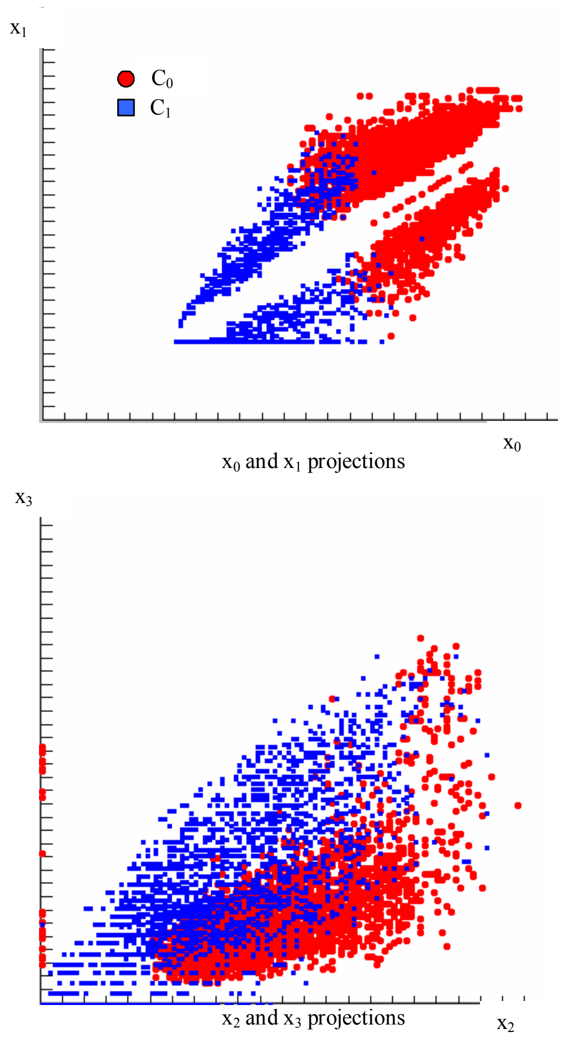


Figure 13: Training set

Table 3: Performance improvement

	C0 (black)	C1 (white)	Global
hyperrectangle error (%)	**0.13**	**14.54**	**1.88**
initial *m*	299	207	506
SVM error (%)	0.93	4.1	1.32
Combination error (%)	**1.19**	**2.84**	**1.39**
m after combination	54	64	118

This example illustrates that it is possible to obtain a decision combining the accuracy of the SVM algorithm and the speed of the hyperrectangles-based method. The final classification error of the hyperrectangles-based method is very close to the SVM result (1.32% for SVM and 1.39% for hyperrectangle). The final hardware cost for this process is λ_H=236 slices, whereas the cost of implementation for SVM is very high, since a total of 2500 support vectors were found during the training step. In this case, the hardware cost of a full parallel decision step is λ_{svm}=662508 !

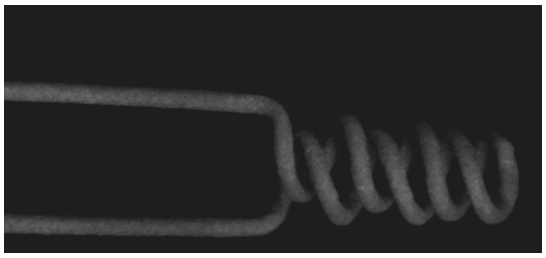
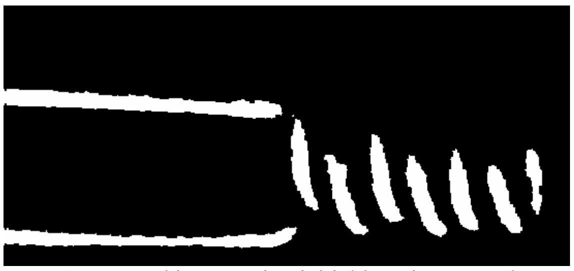
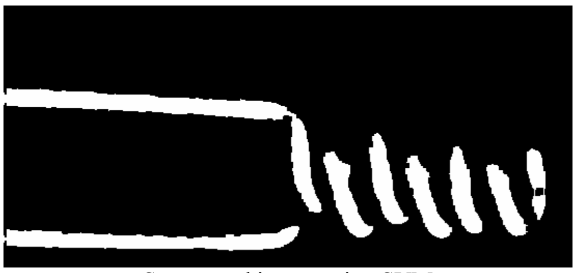


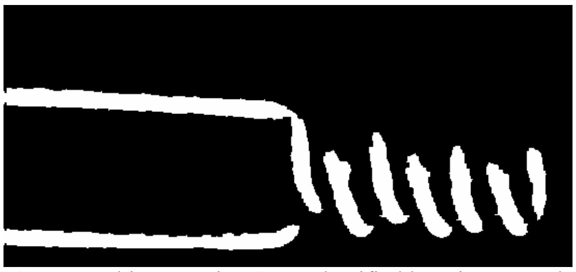
Figure 14: Test image



Segmented image using initial learning set and hyperrectangles



Segmented image using SVM



Segmented image using SVM classified learning set and hyperrectangles

Figure 15: Segmented images

Conclusion

We have shown that it is possible to imitate the performance of the SVM classifier for a lower cost of implementation and increased the accuracy of a particular hyperrectangles-based classifier.

The performance improvement of the basic method is valid in terms of classification as well as integration cost (i.e. speed), since the final number of hyperrectangles is minimised.

We demonstrated the improvement both on Gaussian distributions and in a practical case of textured image segmentation. We developed the whole implementation process, from the learning set definition to FPGA implementation using automatic VHDL generation.

Note that this combination models well our behaviour before an artificial vision quality control problem: the very first decision given by the expert is often modified for limit cases after observation of the results. This can be seen also as a particular application of ambiguities reject method used in many classification algorithms.

Our future work will be the whole implementation of the anomalies detection process on the part presented here in order to illustrate the advantage of the learning steps combination.

Acknowledgements

The authors wish to thank F. Meriaudeau for his help with the English translation of this paper.

References

[1] R. O. Duda and P.E. Hart (1973) *Pattern classification and scene analysis*, Wiley, New York, pp. 230-243.

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

[3] V. Vapnik (1995) *The nature of statistical learning theory* , Springer-Verlag, New York.

[4] P. Niyogi, C. Burges, P. Ramesh (1999) Distinctive Feature Detection Using Support Vector Machines, ICASSP 99, 1: 425-428.

[5] B. Schölkopf, A. Smola, K.-R. Müller, C. J. C. Burges and V. Vapnik (1998) Support Vector methods in learning and feature extraction, *Australian Journal of Intelligent Information Processing Systems,* 1: 3-9.

[6] K. Jonsson, J. Kittler, Y. P. Li, and J. Matas (1999) Support Vector Machines for Face Authentication. In T. Pridmore and D. Elliman, editors, British Machine Vision Conference, pp 543-553.

[7] J. Mitéran, P. Gorria and M. Robert (1994) Classification géométrique par polytopes de contraintes. Performances et intégration , *Traitement du Signal*, Vol 11 : 393-408.

[8] D. Wettschereck and T. Dietterich (1995) An Experimental Comparison of the Nearest-Neighbor and Nearest-Hyperrectangle Algorithms, *Machine Learning,* Vol 19, 1: 5-27.

[9] S. Salzberg (1991) A nearest hyperrectangle learning method. *Machine Learning*, 6: 251-276.

[10] M. Robert, P. Gorria, J. Mitéran, S. Turgis (1994) Architectures for real-time classification processor, *Custom Integrated Circuit Conference*, San Diego CA, pp 197-200.

[11] J. Mitéran, J. P. Zimmer, F. Yang, M. Paindavoine (2001) Access control : adaptation and real-time implantation of a face recognition method, *Optical Engineering*, 40(4): 586-593.

[12] B. Dubuisson *Diagnostic et reconnaissance des formes*, HERMES, Paris, 1990.

[13] J. Mitéran, P. Geveaux, R. Bailly and P. Gorria, Real-time defect detection using image segmentation (1997) *Proceedings of IEEE-ISIE 97*, Guimares, Portugal, pp. 713-716.

[14] R. Enzler, T. Jeger, D. Cottet, and G. Tröster (2000) 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

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

[16] M. A. Hearst, B. Schölkopf, S. Dumais, E. Osuna, and J. Platt (1998) Trends and Controversies - Support Vector Machines. *IEEE Intelligent Systems*, 13(4) : 18-28.

[17] K. Chapman (1996) Constant coefficient multipliers for the XC4000E. Xilinx Application, Note XAPP054, Xilinx, Inc.

[18] J. Kittler, M. Hatef, R. P. W. Duin, J. Matas (1998) On combining classifiers in *IEEE transactions on*

Pattern Analysis and Machine Intelligence, 20(3): 226-239.

[19] B. Moobed (1996) Combinaison de classifieurs, une nouvelle approche, PhD. thesis, Laboratoire d'informatique de polytechnique d'Orsay ; France.

[20] J. Kittler (1978) Feature set search algorithms, *Pattern Recognition and Signal Processing*, Sijthoff and Noordhoff, Alphen aan den Rijn, Netherlands, pp 41-60.

[21] P. Somol, P. Pudil, J. Novovocova, P. Paclik (1999) Adaptative floating search methods in feature selection, *Pattern Recognition Letters*, 20: 1157-1163.