Implantation automatique et optimisée d'une fonction de décision a l'aide des SVM et du VHDL

J. Mitéran, S. Bouillant, M. Paindavoine, E. Bourennane

Le2i - UMR CNRS 5158 Aile des Sciences de l'ingénieur
Université de Bourgogne BP 47870
21078 Dijon - FRANCE
miteranj@u-bourgogne.fr

Résumé

Nous proposons une méthode d'implantation automatique et optimisée d'une fonction de décision particulière. Nous utilisons une combinaison de méthodes basées sur un classifieur fort (SVM) et sur un classifieur faible à base d'hyperrectangles, ainsi qu'un algorithme itératif qui permet de minimiser le coût d'implantation matérielle tout en maîtrisant l'erreur de classification. Nous rappelons les principes des deux méthodes de classification et nous évaluons leurs coûts d'implantation matérielle respectifs. Nous présentons des résultats obtenus à partir de distributions standard provenant de bases de données UCI et d'un exemple de segmentation d'images industrielles. Notre méthode peut être vue comme une approximation d'un classifieur fort en vue d'une implantation massivement parallèle.

Mots clefs

Hyperrectangle, FPGA, classification, segmentation, temps-réel, SVM

Abstract

We propose a method of real-time implementation of an approximation of the support vector machine decision rule. The method uses an improvement of a supervised classification method based on hyperrectangles, which is useful for real-time image segmentation. We use a combination of classification methods: a support vector machine is used during a pre-processing step. We recall the principles of the classification methods and we evaluate the hardware implementation cost of each method. We present our learning step combination algorithm and results obtained using standard distributions and an example of image segmentation coming from a part of an industrial inspection problem.

Keywords

Hyperrectangle, FPGA, classification, segmentation, real-time, SVM

1 Introduction

Nous présentons dans cet article une méthode d'implantation automatique et optimisée d'une fonction de décision particulière. Nous nous intéressons ici essentiellement aux décisions extrêmement rapides (de 5 à 10 ns par décision, donc que nous devons obtenir en quelques cycles d'horloge), ce qui peut être utilisé pour résoudre le problème de la segmentation d'images de haute résolution, basée sur la classification de pixels, ou encore le problème plus général de la reconnaissance de formes dans des bases de données d'images de très grande taille. Nous montrons qu'une combinaison de classifieurs utilisée avec un algorithme d'optimisation et l'avantage offert par les architectures reconfigurables, permet une implantation matérielle efficace tout en conservant de très bonnes performances de classification.

La classification est un problème central de la reconnaissance de formes [3] et de nombreuses approches ont été proposées pour résoudre ce problème, comme les réseaux de neurones [2], les Support Vector Machines (SVM) [19], les K plus proches voisins (Knn) et les méthodes basées sur des noyaux, pour ne citer que les plus communes. De nombreuses implantations matérielles de classifieurs particuliers ont été proposées, pour la plupart basées sur des réseaux de neurones, comme dans [8], [4]. Pourtant, l'implantation d'un classifieur particulier n'est pas souvent optimum en termes de surface de composant utilisée ou de performances de classification, à cause de la structure générale de l'algorithme choisi. Beuchat a proposé des outils pour l'approximation automatique de polynômes dans les FPGA [1], mais l'originalité de notre approximation vient du fait que nous supposons inconnue la forme analytique de la fonction de décision à intégrer. Nous proposons donc une méthode de génération automatique de la description d'une fonction de décision approximant un classifieur fort (BC), qui peut être par exemple un réseau de neurones, une fonction de décision basée sur les Support Vector Machine, ou sur les Knn, etc. Dans des articles précédents [9], [14], nous avons montré qu'il est possible d'implanter un classifieur basé sur des hyperrectangles dans un composant parallèle, dans le but d'obtenir des vitesses de décision très élevées, et dans un autre article [10], nous avons montré que ses performances sont suffisantes pour l'utiliser dans un ensemble de reconnaissance de visages. Dans la première partie de cet article, nous présenterons le problème de minimisation à mettre en place. Dans la seconde partie, nous décrirons la méthode proposée, basée sur l'hypothèse qu'une combinaison de cet algorithme avec le classifieur BC et

l'ajout d'une phase de régularisation permet d'obtenir de bonnes performances à la fois en termes de surface de composant utilisée et en termes de performances de classification. Dans la troisième partie, nous présentons des résultats issus d'une part de distributions gaussiennes, et d'autre part issus de basées de données standard provenant d'UCI [12], avant de donner un exemple de segmentation d'images issues d'un problème industriel.

2 Méthode proposée

2.1 Présentation du problème

L'idée de base d'une implantation générale d'une fonction de décision provient du fait que dans un espace discret et borné (typiquement basé sur des octets dans la plupart des implantations matérielles), la frontière issue d'un classifieur est localement linéaire et parallèle aux axes de l'espace des paramètres. Cette constatation nous permet de conclure qu'il est possible de classer un échantillon en comparant chacune de ses composantes avec un ensemble de simples seuils et en utilisant une combinaison logique des résultats de ces comparaisons (aucune multiplication ou addition n'est ici nécessaire). La règle de classification est donc dans ce cas une simple combinaison logique de comparaisons, approximant la fonction de décision réelle, qui est supposée ici connue à travers le classifieur fort BC. Les arbres de décision sont basés sur ce modèle, mais sont essentiellement séquentiels, et une famille de classifieurs parallèles a été définie, basée sur l'algorithme NGE décrit par Salzberg [15], dont les performances ont été comparées aux Knn par Wettschereck and Dietterich [20]. Cette méthode consiste à diviser l'espace des paramètres en un ensemble d'hyperrectangles, pour lesquels de simples comparateurs permettent de vérifier la condition d'appartenance.

Chaque échantillon est défini par un vecteur de paramètres **x** dans un espace de dimension D et par sa classe C(**x**)=*y*:

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

L'appartenance de la composante x_k à un intervalle donné I_{ik} ($i^{ème}$ hyperrectangle et $k^{ème}$ paramètre) est vérifiée par la réalisation des deux conditions suivantes: ($x_k > a_{ik}$) et ($x_k < b_{ik}$), où a_{ik} et b_{ik} sont respectivement les bornes inférieures et supérieure de chaque hyperrectangle. La vérification de l'appartenance d'un échantillon inconnu **x** à une classe *y* résulte donc d'un ensemble de comparaisons qui peuvent être menées en parallèle pour chaque paramètre et pour chaque hyperrectangle de chaque classe *y*. La fonction de décision est donc :

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

m_y est le nombre d'hyperrectangles definissant la classe y. Le nombre total d'hyperrectangles est :

$$\Phi = \sum_{y=1}^{\Omega-1} m_y ,$$

où Ω est le nombre de classes. Les sommes et produits sont les opérateurs logiques. Les performances de notre propre implantation parallèle ont été étudiées dans [9].

Notre but dans cette étude est de trouver un ensemble d'hyperrectangles H_{opt} définissant une approximation de la fonction de décision du classifieur BC, dont l'erreur de classification est supposée être e_{BC}. La solution n'est pas unique, et les performances finales de classification dépendent de Φ. Nous devons donc minimiser l'erreur du classifieur basée sur les hyperrectangles e_H tout en minimisant le coût d'implantation matériel λ_H et en maximisant la vitesse de décision (cette dernière maximisation étant réalisée via le parallélisme intrinsèque à la décision basée sur les hyperrectangles). Le problème peut être résumé de la manière suivante:

Trouver $H_{opt} = \{H_1, H_2, ..., H_\Phi\} / e_H \le e_{BC} + \xi, \lambda_H \le \lambda_{\max}$,

où $\lambda_{\max}$ est un seuil choisi par l'utilisateur dépendant du composant cible et où ξ représente la qualité d'approximation.

Nous avons estimé λ_H en considérant comme cible matérielle les Field Programmable Gate Array (FPGA). Ces dernières années, les FPGAs sont devenus de plus en plus importants et puissants et ont trouvé leur place dans la conception des systèmes. Les FPGA sont utilisés durant le développement, le prototypage et peuvent être remplacé par des ASICs pour la production de volume [4]. L'avantage de ces composants réside principalement dans leur reconfigurabilité [5]. Il est notamment possible d'intégrer les constantes (les limites des hyperrectangles) dans l'architecture de la fonction de décision. La structure élémentaire du FPGA de la famille Virtex (Xilinx) est le slice (qui contient deux LUT de 16 bits), et un composant peut en contenir quelques milliers. Etant donné qu'une règle de décision demande deux comparateurs par hyperrectangle et par paramètre, nous évaluons λ_H, (nombre de slices nécessaires à l'implantation) avec:

$$\lambda_H = D\sum_{y=1}^{\Omega-1} m_y = D\Phi \quad (2)$$

Cette estimation est possible car le résultat binaire L_B d'une comparaison entre une variable A et une constante B est une fonction F_B des bits de A:

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

Nous pouvons écrire L_B de la manière suivante (pour tout octet B tel que 0<B<255):

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

L'opérateur $\oplus$ étant soit l'opérateur AND soit l'opérateur OR, en fonction de la position de $\oplus$ et de la valeur de B. Dans le pire des cas, la structure particulière de L_B peut être stockée dans deux Look Up Tables (LUT) de 16 bits chacune et mises en cascade (soit un slice au total). En moyenne et compte tenu des simplifications possibles pour de nombreuses constantes, nous obtenons 0,5 slices par comparaison. Nous avons ainsi développé un outil qui génère automatiquement la description VHDL d'une fonction de décision après l'étape d'apprentissage (c'est-à-dire connaissant les valeurs de bornes des hyperrectangles). Nous utilisons ensuite un synthétiseur VHDL standard pour générer l'implantation matérielle finale dans le FPGA.

L'estimation de λ_H et de e_H sera obtenue en découpant l'ensemble des données en un ensemble d'apprentissage *S* et un ensemble de test *T*. La dimension D de l'espace étant constante, il est clair d'après (2) que nous devons minimiser Φ afin de respecter la contrainte $\lambda_H \leq \lambda_{\max}$. Nous avons donc défini une stratégie permettant d'obtenir H_{opt}.

2.2 Détermination des Hyperrectangles : méthode de base

Le coeur de la stratégie employée est la détermination d'un ensemble d'hyperrectangles *H* à partir d'un ensemble d'échantillons S.

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

Pendant la première étape, un hyperrectangle $H(\mathbf{x})$ est construit pour chaque échantillon **x** de de la manière suivante : chaque quadrant Q_t (voir Fig. 1) défini un espace où $d_\infty(\mathbf{x}_k, \mathbf{x}_l) = \left| x_t^k - x_t^l \right|$ avec

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

Nous déterminons **z**, le plus proche voisin appartenant à une classe différente dans chaque quadrant Q_p. Si d_p est la distance entre **x** et **z** dans un quadrant donné Q_p, la limite de l'hyperrectangle est $d_f = d_p.R_p$. Le paramètre R_p doit être inférieur ou égal à 0,5. Cette contrainte permet d'assurer que l'hyperrectangle ne peut contenir aucun échantillon d'une classe opposée.

Pendant une deuxième phase, les hyperrectangles d'une classe donnée sont fusionnés afin de minimiser leur nombre [9]. Nous avons évalué les performances de cet algorithme dans plusieurs cas, aussi bien à partir de distributions théoriques qu'à partir de cas réels [10]. Nous avons comparés les performances avec des réseaux de neurones, les Knn et une méthode basée sur le noyau de Parzen [3]. Il est clair que l'algorithme est moins performant que les autres méthodes lorsque les distances interclasses sont trop faibles : un nombre important d'hyperrectangles est créé dans la zone de recouvrement entre les classes, augmentant le coût d'implantation matérielle et dégradant les performances (sur-apprentissage).

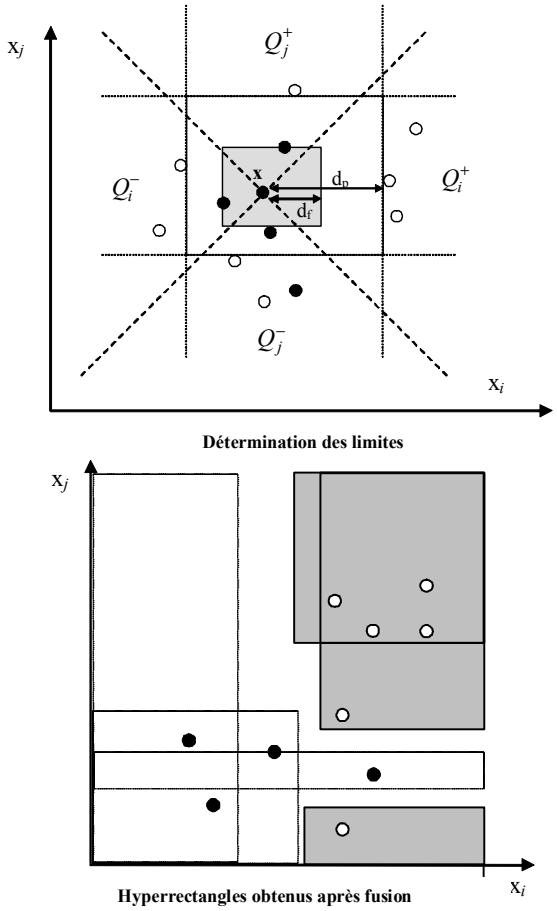


Fig. 1 **Détermination des Hyperrectangles**

2.3 Réduction du sur-apprentissage

Une des causes de ce sur-apprentissage est la présence d'éléments ambigus dans la base d'apprentissage *S*. Pour rejeter ces éléments, une des méthodes les plus utilisées consiste à éditer les échantillons en les classant à l'aide de la règle des Knn. Cette méthode peut également être vue comme une combinaison de méthodes d'apprentissage [13]. Nous avons étendu ce principe, en choisissant le classifieur BC pour la phase d'édition. Or il a été prouvé dans la littérature qu'un des avantages des SVM par rapport aux autres algorithmes qu'il peut être analysé à partir des concepts théorique de l'apprentissage, tout en offrant de très bonnes performances dans des cas réels [16], [7], [6]. Dans les paragraphes suivants, nous avons donc choisi cette méthode, ainsi que la règle des Knn, comme étant les classifieurs forts que nous souhaitions approximer et implanter. Tout autre classifieur pourrait être utilisé de la même manière.

Rappel du principe des SVM

Les Support Vector Machine (SVM) ont été introduites par Vladimir Vapnik [19] à la fin des années 80. Nous

rappellerons ici essentiellement la fonction de décision, la phase d'apprentissage étant largement décrite dans la littérature.

Le principe des SVM est de passer depuis l'espace initial R_D dans un espace de dimension élevé Q; le passage étant assuré par une fonction noyau K. La méthode permet de trouver une séparation linéaire optimum entre les classes dans l'espace Q. Cette séparation maximise la marge entre les échantillons placés dans l'espace Q et elle-même. La frontière est déterminée en résolvant le problème quadratique suivant: maximise

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

En suivant les contraintes $\sum_{i=1}^{n} \alpha_i . y_i = 0$ and $0 \le \alpha_i \le T$ pour i=1, 2, …, n où $x_i \in R_d$ sont les échantillons d'apprentissage et $y_i \in \{-1,+1\}$ la classe correspondante. T est une constante nécessaire dans les cas où les classes ne sont pas linéairement séparables. $K(\mathbf{u}, \mathbf{v})$ est un produit scalaire dans l'espace intermédiaire Q et peut être défini comme une fonction noyau dans l'espace d'entrée. Le choix du noyau $K(\mathbf{u}, \mathbf{v})$ détermine la structure de l'espace intermédiaire Q. Vapnik présente dans [4] trois types de SVM: polynomiale, Radial Basis Function (RBF) et Perceptron multi couche. Nous avons ici choisi le noyau RBF:

$$K(\mathbf{x}, \mathbf{y}) = e^{\left(\frac{-\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}\right)} \quad (4)$$

Les autres paramètres de la fonction de décision (5) sont calculés à partir de (3), c'est à dire l'ensemble des paramètres $\{\alpha_i\}_1^n$ qui déterminent les vecteurs supports et le scalaire b. L'hyperplan séparateur est construit à partir des vecteurs pour lesquels $\alpha_i \neq 0$. Ces vecteurs sont dits les *vecteurs supports* et sont à la marge de la frontière. Le nombre Ns de vecteurs supports détermine la précision et la vitesse de la phase de décision. Dans l'espace d'entrée R_d, la règle de décision est alors :

$$\mathrm{C}(\mathbf{x}) = \mathrm{Sgn}\left(\sum_{i=1}^{Ns} y_i \alpha_i \cdot K(\mathbf{s}_i, \mathbf{x}) + b \right) \quad (5)$$

Où les $\boldsymbol{s}_i$ sont les vecteurs supports définis pendant la phase d'apprentissage. Il a été prouvé dans la littérature que les SVM donnent de bons résultants dans de nombreux cas pratiques mais le coût d'une implantation matérielle complètement parallèle dans un ASIC ou un FPGA est beaucoup plus important que dans le cas de la méthode basée sur des hyperrectangles. Même si l'on peut tabuler la fonction exponentielle, le produit scalaire K demande des multiplications et des additions. Ainsi la fonction finale demande au moins une multiplication et une addition par vecteur support. Nous avons évalué λ_{SVM}, le coût d'implantation matériel des SVM comme étant [11]:

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

Combinaison

La méthode de combinaison permettant l'édition de la base d'apprentissage puis l'approximation de BC par les hyperrectangles est :

- A partir d'un ensemble d'apprentissage S

$S = \left\{ (\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_p, y_p) \right\}$, construire un modèle M à l'aide de BC. Dans le cas des SVM :

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

- construire **S'**, le nouvel ensemble d'apprentissage, en classant chaque élément de S à partir de M et suivant l'équation (4) :

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

- Construire H, ensemble d'hyperrectangles, à partir de S' et de l'algorithme décrit ci-dessus.

2.4 Stratégie appliquée pour minimiser le coût matériel et l'erreur de classification

Etant donné que le coût matériel dépend du nombre d'hyperrectangles, qui lui-même dépend de p, nombre d'échantillons de l'ensemble d'apprentissage S, la stratégie la plus simple est d'itérer la méthode précédente, en augmentant p jusqu'à ce que le but soit atteint en termes de performances de classification ($e_H \le e_{BC} + \xi$) et ce tant que $\lambda_H \le \lambda_{\max}$.

Si le nombre d'échantillons de S n'est pas suffisant pour obtenir de bonnes performances, il est possible de l'augmenter artificiellement en ajoutant une perturbation ou vecteurs aléatoires, puisque ceux-ci seront classés par BC. Ces échantillons permettront au classifieur faible d'adopter le pouvoir de généralisation du classifieur fort. Cette méthode peut être vue comme une phase de régularisation améliorant l'approximation de BC (Fig. 2).
Ainsi, pour chaque échantillon **x** de S', nous générons un sous-ensemble B de q éléments:

$$B = \left\{ (\mathbf{x}"_1, y"_1), (\mathbf{x}"_2, y"_2), ..., (\mathbf{x}"_q, y"_q) \right\}$$

où x_{ik}"= $x_{ik} + \varepsilon_k$, et ε_k est une valeur aléatoire définie dans un intervalle [-δ, δ], pour chaque paramètre k , et δ est fixé par l'utilisateur. La classe y" de chaque nouvel échantillon est déterminé à partir de BC et du modèle M. Le nombre total d'échantillon de S" est p"=p+q.p. Augmenter ainsi le nombre total d'échantillons améliore la précision de l'approximation. Il est clair qu'un plus grand

nombre d'hyperrectangles approximera mieux la surface de décision qu'un petit nombre d'entre eux. Afin de trouver le meilleur compromis entre le coût matériel et les performances de classification, nous itérons la méthode décrite dans le paragraphe précédent en utilisant des sous-ensembles de *S''*. Le processus peut être arrêté si l'erreur de classification a atteint son minimum ou si le nombre maximum de slices (λ_{max}) est atteint.

L'algorithme est donc :

0-Depuis un ensemble de données, construire *S* (ensemble d'apprentissage) et *T* (ensemble de test),

1-depuis *S*, construire un modèle M à l'aide de BC,

2-estimer l'erreur e_{BC} à partir de M et de *T*,

3-construire *B*, ensemble de perturbations,

4-construire *S'*, en classant chaque élément de *S* à partir de M,

5-construire *B'*, en classant chaque élément de *B* à partir de M,

6-construire $S'' = S' \cup B'$,

7-construire *X*, en sous échantillonnant *S''* avec q' échantillons,

8-construire *H*, ensemble d'hyperrectangles, à partir de *X*,

9-estimer l'erreur de classification e_H à partie de *H* et *T*,

10-estimer λ_H,

11-Arrêter si $\lambda_H > \lambda_{max}$ ou if $e_H \leq e_{BC} + \xi$, sinon augmenter q' et aller en 7,

12-utiliser le dernier ensemble *H* comme ensemble final H_{opt}, et générer la description VHDL correspondante,

13-implanter le module VHDL dans un design à base de FPGA.

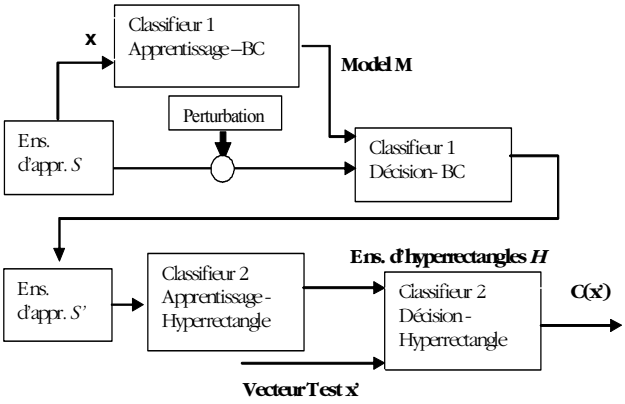


Fig. 2 **Combinaison des phases d'apprentissage**

3 RESULTATS

3.1 Validation expérimentale à l'aide de distributions gaussiennes

Nous avons dans un premier temps et à titre purement illustratif validé le principe de la méthode décrite ci-dessus à partir de distributions gaussiennes multimodales. La configuration testée contient 3 classes dans un espace à dimension 2. Un exemple d'approximation des SVM avec noyau RBF (eq. 4) et de règles des Knn est représenté Fig. 4.

L'erreur de classification e_H en fonction de q' est reportée Fig. 3. Etant donné que e_{SVM}=9.17%, nous obtenons pour q'=75, λ_H =35 slices avec e_H=9,75%, en choisissant ξ=1. Cette expérience illustre bien le fait que la méthode basée sur les hyperrectangles a imité la fonction de décision des SVM (ainsi que celle des Knn), donnant une bonne approximation des frontières, avec un coût d'implantation très faible.

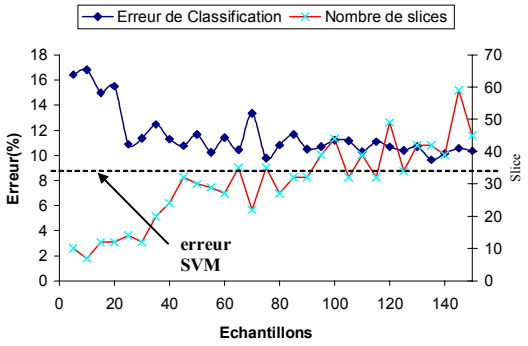
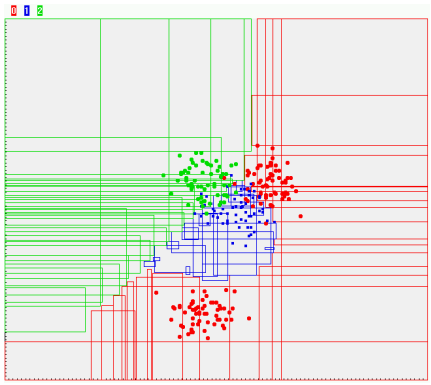
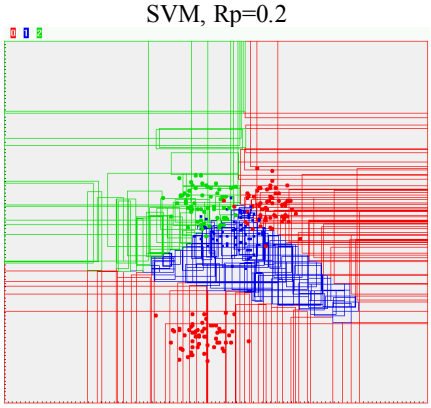


Fig. 3 Résultats avec BC=SVM, R_p=0.2



SVM, Rp=0.2



Knn, Rp=0.5, k=15

Fig. 4 Approximation des frontières des classifieurs

Il est à noter que le coût d'une implantation matérielle complètement parallèle des SVM serait ici, avec *Ns*=338, de λ_{SVM} =72944 slices. Il est toutefois clair qu'une mémorisation complète du résultat des SVM serait possible dans

ce cas particulier de la dimension 2. Nous avons donc appliqué la méthode à des espaces de dimension plus importante.

3.4 Validation expérimentale à partir de bases de données standard

Nous avons appliqué notre méthode à diverses données provenant de UCI [12]. La première base testée est celle de segmentation d'images, à 4 classes (Window, Cement, Grass et Path). La dimension originale de l'espace est D=18 mais a été réduite à D=5 à l'aide d'une méthode de sélection de paramètres [17]. L'erreur de classification (voir Fig. 5) des SVM est e_{SVM}=2.75 %, et le résultat obtenu par approximation est de e_H=2.80%, pour un coût de λ_H=900 slices. Dans ce cas, une implantation des SVM aurait donné λ_{SVM}=76616 slices (*Ns*=76).

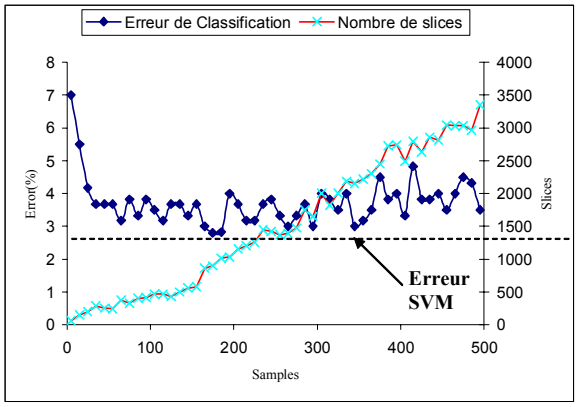


Fig. 5 **Résultats à partir de la base "image"**

Nous avons appliqué également la méthode sur les données des vins (« wine ») et des iris, qui contiennent 3 classes. Nous avons utilisé dans chaque cas les SVM et la règle des Knn comme classifieur fort. Les résultats sont reportés Table 1.

Table 1 : Résultats obtenus à partir de données réelles

BC	Data-base	E_{BC} (%)	e_h (%)		λ_H	
			Rp=0.2	Rp=0.5	Rp=0.2	Rp=0.5
SVM (RBF)	Image	3.33	3.17	2.50	45	751
	Wine	7.78	7.78	5.56	620	1097
	Iris	2.67	2.67	2.67	71	190
Knn (k=15)	Image	2.83	3.67	2.33	190	900
	Wine	6.67	6.67	5.56	680	2240
	Iris	2.67	2.67	2.67	80	470

La base de donnée "wine" contient 3 classes, et la dimension de l'espace a été réduite à D=4 par sélection de paramètres. La base de donnée Iris n'est pas vraiment adaptée à notre méthode car elle ne contient que très peu d'échantillons : la base de test est donc très pauvre. Toutefois, on peut noter que dans tous les cas notre méthode donne une erreur de classification au moins égale à celle des autres classifieurs testés, et ce pour un coût d'implantation parallèle relativement faible. La phase de fusion des hyperrectangles, qui est efficace dès lors que Rp<0,5, permet également de réduire ce coût.

3.5 Application à la segmentation d'image temps réel pour la détection de défauts

Nous avons appliqué la méthode lors d'un prétraitement nécessaire à la détection de défauts sur des pièces industrielles (Fig. 6). Ce prétraitement consiste à distinguer partie interne et partie externe de chaque spire, afin de pouvoir ensuite effectuer un certain nombre de mesures qui permettront de classer les pièces comme bonnes ou défectueuses. La spire étant texturée, un simple seuil ne pourrait être assez robuste pour segmenter ainsi l'image. Nous avons défini un ensembles de paramètres caractéristiques des textures, tout en gardant à l'esprit les contraintes de traitement temps réel. En effet, la taille des images est de 1288x1080 pixels, et la vitesse d'acquisition est de 10 images/s, ce qui nous oblige à réaliser le prétraitement en moins de 72 ns. Nous avons ainsi sélectionné les quatre paramètres suivants :

x_0 moyenne de la luminance dans des fenêtres 8x8,

x_1 est la nouvelle valeur du pixel après égalisation d'histogramme,

x_2 moyenne d'un filtre de Sobel dans des fenêtres 8x8,

x_3 moyenne d'un contraste local dans des fenêtres 8x8.

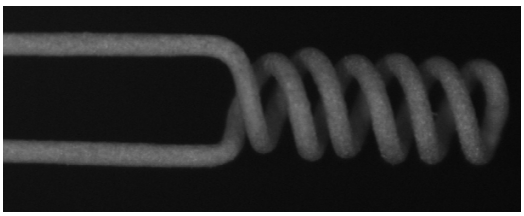


Fig. 6: Pièce à contrôler.

Nous avons défini les classes de l'ensemble d'apprentissage S manuellement, à partir de quelques images. Pour chaque classe, p=5000 pixels ont ainsi été choisis de manière aléatoire dans les parties interne et externe des spires. Nous avons représenté deux projections de cet ensemble Fig. 7. Nous avons appliqué la méthode décrite précédemment sur 10 images de test. Un de ces exemples est reporté Fig. 9. Les résultats en termes de classification et de coût d'intégration sont reportés Fig. 8.

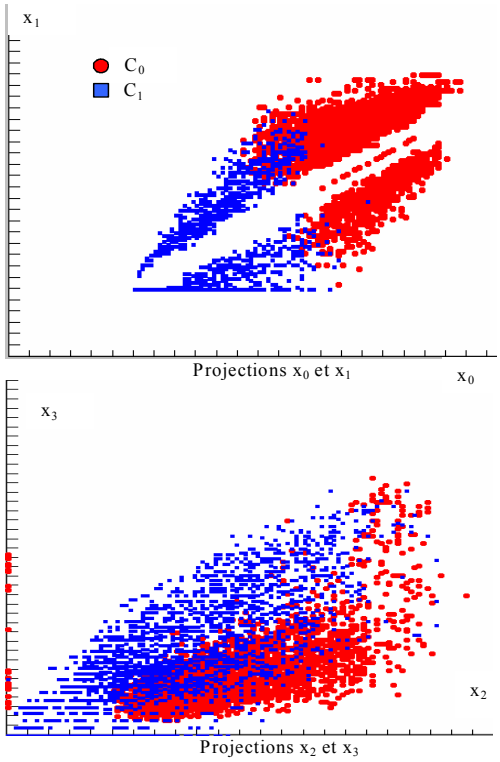


Fig. 7 Ensemble d'apprentissage

Cet exemple, comme les précédents, illustre bien le fait que l'erreur de la combinaison converge vers celle des SVM. Le résultat final est de 1.94% pour les SVM et de 2.20% pour les hyperrectangles, et ce pour un coût d'implantation bien moins élevé dans le cas des hyperrectangles. En effet, nous avons dans ce cas λ_H=2110 slices, contre λ_{svm}=376 208 slices pour les SVM.

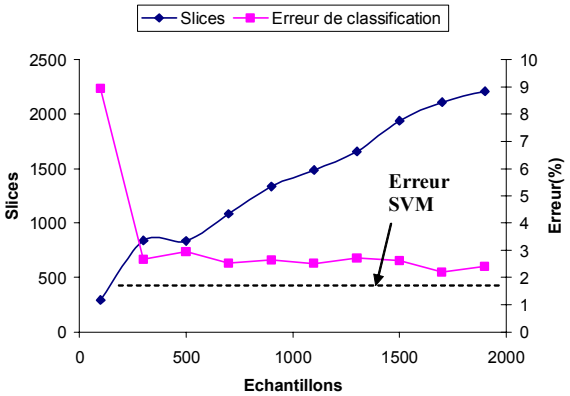


Fig. 8 Performances

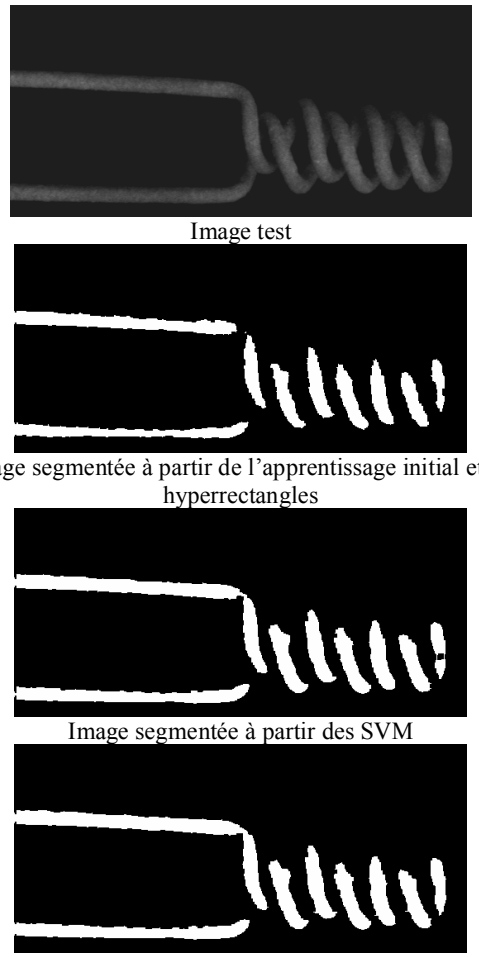


Fig. 9 Images Segmentées

4 Conclusion

Nous avons développé une méthode permettant la génération automatique d'une implantation matérielle d'une fonction de décision particulière. Cette méthode peut être utilisée dans de nombreuses tâches de reconnaissance de formes, comme la segmentation d'image. L'utilisateur peut choisir de donner la priorité au coût matériel en fixant λ_{max} et en réglant ξ, ou de donner la priorité aux performances de classification en fixant ξ très faible et en laissant λ_{max} élevé.

Nous avons validé la méthode de manière expérimentale, sur des distributions théoriques aussi bien que sur des distributions réelles. Nous avons montré qu'il est possible de trouver un coût d'implantation faible pour une erreur de classification qui converge vers le classifieur BC. Nous avons développé l'intégralité du procédé, depuis la définition de l'ensemble d'apprentissage jusqu'à la génération automatique du VHDL correspondant.

References

[1] J.L. Beuchat and E. Sanchez, Using On-Line Arithmetic and Reconfiguration for Neuroprocessor Implementation. In José Mira and Juan V. Sánchez-Andrés, editors, IWANN 99, Engineering Applications of Bio-Inspired Artificial Neural Networks, Springer, pp 129-138, 1999.

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

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

[4] 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.

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

[6] 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.

[7] 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.

[8] 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

[9] 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.

[10] 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.

[11] J. Miteran, S. Bouillant, E. Bourennane, SVM approximation for real-time image segmentation by using an improved hyperrectangles-based method, Real-Time Imaging, Elsevier, 9 (3), pp. 179-188, Juin 2003.

[12] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases. Department of Information and Computer Science, University of California, Irvine, California.

[13] J. Kittler, M. Hatef, R. P. W. Duin, J. Matas, On combining classifiers in *IEEE transactions on pattern analysis and machine intelligence*, 20(3): pp 226-239, 1998.

[14] 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.

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

[16] 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.

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

[18] Y. Taright, M. Hubin, FPGA Implementation of a Multilayer Perceptron Neural Network using VHDL, 4th Int.l Conf. on Signal Processing (ICSP'98), Beijing, october 12-16 1998.,Vol2, pp 1311-1314.

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

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