Exploiting Application Locality to Design Low-Complexity, Highly Performing, and Power-Aware Embedded Classifiers

Cesare Alippi, *Fellow, IEEE,* and Fabio Scotti, *Member, IEEE*

***Abstract*—Temporal and spatial locality of the inputs, i.e., the property allowing a classifier to receive the same samples over time—or samples belonging to a neighborhood—with high probability, can be translated into the design of embedded classifiers. The outcome is a computational complexity and power aware design particularly suitable for implementation. A classifier based on the gated-parallel family has been found particularly suitable for exploiting locality properties: Subclassifiers are generally small, independent each other, and controlled by a master-enabling module granting that only a subclassifier is active at a time, the others being switched off. By exploiting locality properties we obtain classifiers with accuracy comparable with the ones designed without integrating locality but gaining a significant reduction in computational complexity and power consumption.**

***Index Terms*—Application-level design, classifier design, embedded systems, gated-parallel classifiers, power-aware design.**

I. Introduction

EMBEDDED application, wireless sensor networks, and pervasive computing markets force research and development teams to develop highly performing, small, low-power consuming and cheap final systems. In the same direction, ICT engineers are starting addressing the design of low-power applications also on the embedded software side to keep power consumption at a reasonable level.

Computational complexity issues and power-aware policies must be faced to provide an effective embedded system design providing high performance, low execution time, and energy savings. In this paper, we consider applications requiring a computational intelligence-based classification core and an embedded system tailored to it.

Power consumption and computational complexity aspects have been directly/indirectly tackled in classifiers design where the main goal is primarily accuracy maximization, e.g., see [1]–[3]. References [4] and [5] propose a finite precision and wordlength dimensioning for neural network variables, issue that indirectly impacts on power consumption and device complexity reduction. In the same direction, complexity reduction can be achieved through minimization of the network topology at different levels, for instance, through network pruning [6] and dimensioning at connection [7]–[9] and neuron [10] levels. Since complexity is either related to the number of instructions to be executed (execution cycles) or device complexity, complexity reduction allows for energy-saving and fast execution. In this paper we consider classification applications possessing the locality property and show how this *a priori* information can be fruitfully applied to design power and complexity aware classifiers. The locality property requires at least one of the following statements to hold.

1) *Temporal locality*: The probability that pattern x_i presented to the classifier at time t is also given at some close time $t + \tau$, $\tau > 0$ is high.
2) *Spatial locality*: The probability that consecutive patterns presented to the classifier belong to the same neighborhood is high.

The locality property is particularly relevant in those applications bounded to operate in working points (e.g., under the presence of a controller) or characterized by trajectories of the patterns confined in limited regions. Locality properties cannot be exploited by traditional neural classifiers, e.g., feedforward neural classifiers, and require an *ad hoc* design.

To identify the most suitable structure we refer to [1] where classifiers have been classified as: Monolithic (a unique classifier solves the task, e.g., nearest neighborhood classifiers [2], feedforward neural network classifiers [11]), parallel (a set of classifiers are considered and act in parallel; their output is processed by an output grouping module to yield the final output, e.g., gated-parallel classifiers [1], multiple classifiers [1]), cascade or linear (a set of classifiers are activated in sequence, e.g., see [1]) and hierarchical (classifiers are combined in a tree-like structure, e.g., mixtures of experts [12]).

We identified in a variant of the gated-parallel class the classifier structure able to fully exploit locality properties.

The novelty of the paper mainly resides in the formalization and use of locality properties and, to a second extent, in the rather articulated design methodology integrating accuracy and complexity/low-power consumption constraints. Novel is also the features optimization/reduction algorithm (which consists in a variant of existing ones).

The structure of the paper is as follows. The topological structure of the suggested gated-parallel classifier and its design are given in Section II. Section III, by receiving the classifier core topology delineated in section Section II, explores the design space with genetic algorithms to identify the most adequate classification families (subclassifiers composing the gated-parallel structure might be characterized by a different classification paradigm) and provides the target classifier. Finally, results

Manuscript received April 22, 2004; revised June 4, 2005.

C. Alippi is with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, 20133 Milano, Italy (e-mail: alippi@elet.polimi.it).

F. Scotti is with the Department of Information Technologies, University of Milan, 26013 Crema, Italy (e-mail: fscotti@dti.unimi.it).

Digital Object Identifier 10.1109/TNN.2006.872345

are given in Section IV where the methodology has been validated on real applications and benchmarks.

II. Designing Multiplexed Classifiers

A classification based on the gated-parallel philosophy requires decomposition of the problem in subproblems, each subproblem being associated with a subclassifier activated by a master module. Subclassifiers are trained/configured on appropriate subdomains, hence, maximizing their synergy with the local environment. The grouping output module disappears from the canonical gated-parallel structure for complexity reasons while we allow subclassifiers to be different in structure and topology. We name the obtained classifier structure "multiplexed classifier" since the master enabling module acts as a multiplexer activating a subclassifier at a time, the others being switched-off.

Temporal locality can be seen as a particular case of spatial locality; as such, when the property holds patterns will come with high probability from the same subdomain. The subclassifier associated with such subdomain will be generally less complex than the one associated with the whole domain and, in addition, will continuously be active with high probability for some time (meanwhile, all other subclassifiers are switched off). Complexity and power consumption reduction derives from these observations.

The design of a multiplexed classifier requires solution to the following subproblems.

1) *Topological structure identification (multiplexed classifier level)*: Identification of the suitable number of subclassifiers d and partitioning the input-output space in d subdomains, one for each subclassifier;
2) *Topological structure identification (subclassifier level)*: Selection, for each subdomain, of the features set relevant to the specific subclassification problem;
3) *Master module configuration*: Design and configuration of the master enabling module;
4) *Subclassifiers design*: Design of the final multiplexed classifier by selecting and configuring the most suitable models for the d subclassifiers.

Each subproblem contributes to design a performing classifier as well as influences the complexity (power consumption) aspect. Accuracy and complexity are strictly related and generally competing terms, particularly in the design of a monolithic classifier where only steps 3) and 4) are envisaged (and applied to the unique classifier); differently, a multiplexed classifier (and similar classifier structures) introduces additional degrees of freedom in the design space [steps 1)–2)].

To ease and make effective the methodology, we separate the design in two phases aiming at decoupling accuracy from complexity. In the first phase [steps 1)–3)] the focus is primarily on complexity (complexity constraint integration at application level). At this level, complexity reduction is obtained by acting on the number of classifiers and their topological complexity yet leaving enough degrees of freedom for a subsequent integration of the accuracy requirement. Conversely, the second phase, [step 4)] primarily focuses on accuracy maximization by considering adequate models for subclassifiers meanwhile penalising

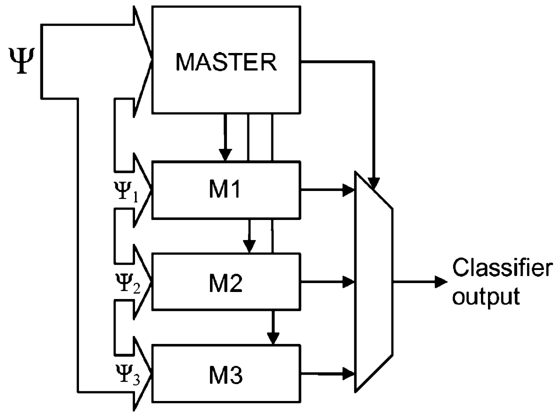

Fig. 1. Three modules multiplexed classifier. Subclassifiers are mutually exclusive and enabled by the master module.

complex solutions. In a way, the envisaged two-phases design partly decouples the complexity issue from the accuracy one by focusing first on complexity at a coarser level (without considering accuracy) and only later requiring accuracy for the classification system (with a penalization of complex solutions). In this section we focus the attention on the first phase; the second phase will be addressed in Section III.

In the following we assume, without loss of generality, that the classifier output is binary with the $\{$*good*, *nogood*$\}$ labels mapped in the $\{0,1\}$ alphabet. As such, a classifier output equal to 1 implies that the process has provided a bad sample. Extension to multivalued classifiers will only trivially affect step 1.

After having selected the number of subclassifiers d, integration of the complexity constraint requests each subclassifier to be optimized in terms of its topological structure at the input/output level (we are neither selecting a model for the subclassifier nor providing a classifier topology at this stage). This can be obtained by identifying Ψ_i, the minimal set of features needed by the ith subclassifier and configuring the master to activate (value one) only a subclassifier at time. Fig. 1 shows a multiplexed classifier with three subclassifiers after application of steps 1)–3). We now detail the operations needed to accomplish steps 1)–3).

A. Domain Partitioning

The problem associated with the determination of the optimal number of subclassifiers can be cast into a data clustering problem where the vicinity concept is augmented with the class label information. If data associated with a classification value are clustered together (spatial locality) it is likely that the cluster will be associated with a subclassifier. Moreover, if the cluster contains only data associated with a label (e.g., the domain is referring to a subdomain where all elements to be classified are good) then the subclassifier degenerates in a module always providing its label once enabled. Of course, in correspondence with a degenerated subclassifier complexity and power consumption are reduced to a minimal value.

Fig. 2 shows the classifier of Fig. 1 where subclassifiers M2 and M3 degenerated to constant values 1 and 0, respectively; we assume that the reference application is a quality analysis problem. Subclassifiers associated with good values (0 value)

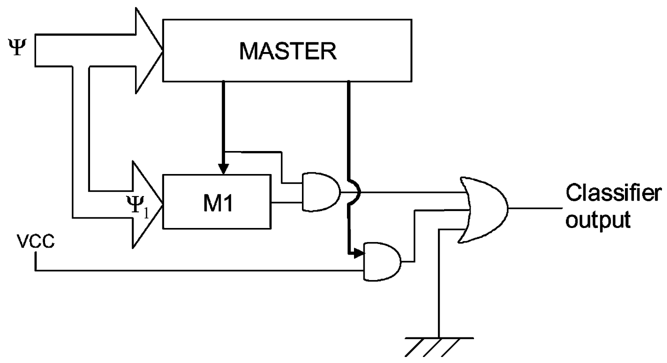

Fig. 2. Three modules multiplexed classifier. Subclassifiers are mutually exclusive and enabled by the master module.

are considered to be the default here since it is likely that the process will maximally provide good samples (then we need only one of such classifiers). All degenerated classifiers are associated with the 0 value line entering the final OR gate. Likewise, degenerated classifiers associated with bad areas provide, once enabled (AND gate), a value 1 to the classifier output (if there are several degenerated classifiers of such type their enabling signals can be OR-ed). The digital nomenclature and design is only for its immediate intuition; we could implement the same functionalities differently, for instance, in a fully software solution.

In the following we will not consider degenerated models for their obvious implementation: The attention will be focused on nondegenerated subclassifiers. Clustering can be carried out by considering the designer's favorite algorithm; here, we consider a supervised Fuzzy C-Means clustering technique [13] for its effectiveness and computational simplicity. The clustering technique divides the samples populating a multidimensional space into d groups by means of an affinity-similarity function; no information about the point labels is considered during the clustering phase. Similarity is calculated with a fuzzy membership function, typically based on the relative distance of the sample with respect to its centre of class. The number of clusters is generally set by the designer; a method for an automatic identification of d based on locality property exploitation is suggested in Section II-D; as an heuristic: d should be increased whenever the new partitioning provides a reduction in the input complexity for nondegenerated classifiers. The rationale behind the heuristic is based on the fact that a subclassifier requiring a reduced number of features is likely to be less complex than the counterpart requiring more inputs. Nevertheless, by increasing d we increase the complexity of the master module. It must be outlined that non degenerated subclassifiers receive a data subdomain containing both label classes and, as such, the critical classification core is decomposed in simpler classification subproblems.

B. Feature Selection

The feature selection step is fundamental in reducing the complexity of the multiplexed classifier since it addresses identification of the minimal number of input features to be presented to each subclassifier (with an obvious impact on the classifier's topology). In addition, features reduction also improves the generalization ability of the classifier since redundant information/degrees of freedom are removed. The interested reader can refer to principal component analysis-like techniques [14]–[16] for relating features reduction with generalization improvement or to [17] for neural-networks based methods.

As it can be expected, a feature selection technique may significantly reduce the features needed to solve a subclassification task; since the subclassifier operates in a subdomain it is likely that only few features will be sufficient to solve the local classification problem. By applying features selection, we obtain an additional significant reduction in complexity.

The designer can consider his/her favorite feature selection method. Here, we suggest a method based on a feature relevance analysis to solve the feature selection problem (see, also, [18]).

The feature extraction problem scales badly with the number of features f since we have to generate 2^f classifiers, each differing in the combination of features: The best feature set is finally the one maximizing accuracy. This obvious comment hides a further problem: Each subclassifier requires a training phase (a time consuming procedure) and a validation phase to test accuracy.

The problem can be solved with an effective heuristics which considers k-Nearest Neighbor (k-NN) classifiers and assumes that their accuracy can be confused with the best Bayes classifier obtainable with the same feature set (i.e., the analysis is optimistic). The advantage resides in the trivial training phase for k-NN classifiers: Given a candidate feature set the classifier can be generated with almost no computational cost [19]. The accuracy of the obtained classifier can be estimated with a leave one out (LOO) validation technique [20] with a correction to deal with the finite number of data [21], [22]. The final feature optimization procedure is summarized in the algorithm given in Fig. 3 where we denoted with Ψ_i a generic feature and with brackets a vector of features.

Intuitively, for each subclassifiers, we are looking for the combination of features maximizing accuracy. Due to the computational complexity of the procedure, instructions ‡ are introduced to limit the number of combinations to be generated (the threshold values are introduced to prune non interesting feature vectors as in a branch and bound approach). If the number of features f is limited we can generate all possible classifiers and select the best performing one according to instruction †.

C. Master Module Construction

The master module can be easily built with a k-NN classifier trained over the d centres of class identified in step II-A. During the operational phase of the multiplexed classifier the pattern to be classified is compared with each centre of class and the closest vector (e.g., according to the Euclidean distance) activates a subclassifier. Selecting a k-NN as a master module is a suitable choice when the number of subclassifiers is not very high (we experienced this is the case in many applications). Conversely, in designs where the number of subclassifiers is high we should opt for a neural network of regression type, e.g., FeedForward Neural Networks (FF) or Radial Basis Functions (RBF) to represent efficiently the master module function.

```
for ( i=1; i<= $d$; i++ ) {        // for each sub-classifier
    Consider the data sub-dominion $D_i$;
    $S=\{[\Psi_1],[\Psi_2],\ldots[\Psi_f]\}$; $\Gamma=\{\}$;
    While (cardinality( $S\neq 1$ ) {
        Generate the $k$-NN classifier on $D_i$ for each element in $S$;
        †Evaluate the LOO performance for each k-NN induced by S;
        Insert the best $k$-NN w.r.t. accuracy in $\Gamma$ ;
        ‡Remove from $S$ those feature vectors whose $k$-NN classifiers show an accuracy below a given threshold;
        $S$ = combinations of all possible union of couples in $S$;
    } Generate the $k$-NN classifiers for all $S\cup\Psi_i$ (not in $S$) and insert in $\Gamma$ ;
    The optimal feature set $\Psi_i$ for the sub-classifier is the one in $\Gamma$ with maximal accuracy;
}
```

Fig. 3. Feature selection algorithm.

When d is high (say above 7) we suggest the designer to test both solutions and select the least complex one.

A k-NN master enabling module based on the Euclidean distance induces a linear partition of the input space and each subdomain becomes a polyhedron (Voronoi partition [23]). When FF or RBF models are considered for the master module the domain boundaries are represented by nonlinear curves instead of segments. Aggregation schemas for subspaces as the ones suggested in [24] and [25] can also be taken into account. More in general, any clustering and aggregation schema can be considered in the methodology provided that there it exists a master module enabling only one submodule at-time.

D. Complexity and Application-Level Properties

In the following, we denote by Φ_{MC} the multiplexed classifier, by Φ_M the best non multiplexed reference classifier and by C the complexity function.

Complexity, estimated by means of the number of processor clock cycles needed to execute the classifier code, has been evaluated with the clock-accurate simulation tool WATTCH developed at Princeton University [26] which, by using the simplescalar Portable ISA (PISA) architecture simulator [27], profiles the compiled code for a generic scalar architecture.

Once the clustering algorithm has partitioned the application space, it is natural to associate an activation probability $P_i=\int_{\Omega_i}p(x)dx$ to a generic subclassifier ($p(x)$ is the application input probability density function and Ω_i its subdomain). In other words P_i denotes the probability that a pattern to be classified belongs to the ith subclassifier domain of complexity C_i. If we denote by C_{Master} the complexity of the master module, then the effective complexity of the multiplexed classifier becomes

$$C(\Phi_{MC})=C_{\text{Master}}+\sum_{i=1}^{d}P_iC_i. \tag{1}$$

The unknown probability can now be estimated by using the available activation frequencies coming from the application profiler. To this end we can use all available N data to generate an estimate of the probability needed in (1). If the ith subclassifier is activated N_i times out of N then the classifier complexity can be expressed as

$$C(\Phi_{MC})=C_{\text{Master}}+\sum_{i=1}^{d}\frac{N_i}{N}C_i. \tag{2}$$

A lemma can be derived which states that if the complexity of each subclassifier (master enabling module for the classifier plus subclassifier complexity) is smaller than the reference classifier one then the complexity of the resulting multiplexed classifier Φ_M is smaller than that of the reference classifier.

Lemma: If $C_i+C_{\text{Master}}<C(\Phi_M),\forall i,i=1,d$ then $C(\Phi_{MC})<C(\Phi_M)$. The proof immediately follows by noting that

$$C(\Phi_{MC})=C_{\text{Master}}+\sum_{i=1}^{d}P_iC_i<C(\Phi_{MC})\sum_{i=1}^{d}P_i=C(\Phi_M) \tag{3}$$

holds for any application. It seems from experimental evidence that the hypothesis required by the lemma is almost always satisfied in nontrivial classification problems.

It is easy to extend locality properties proper of caching mechanisms to multiplexed classifiers if the application possesses the locality property. In fact, at the multiplexed classifier level we can define two types of locality at the subclassifier level.

- *Temporal locality*: It is high the probability that if the ith subclassifier is active at time t it is also active at time $t+1$. To evaluate the temporal locality at the multiplexed classifier level we generate the $T_i(\tau)$ curve which provides, for different values of τ, the probability that the ith classifier active at time t is continuously active up to time $t+\tau$. When inputs are independent and identically distributed we have that $T_i(\tau)=P_i^\tau$: If a subclassifier is characterized by a high activation probability then the temporal locality property can be relevant.
- *Sequential locality*: If the ith classifier is active at time t it is high the probability the same will be active at some close time $t+\tau$.

By exploiting the $T_i(\tau)$ curves we can derive an automatic procedure for selecting the number of subclassifiers d. Since it is our interest to maximize temporal locality we can select the optimal d as the one maximizing the ensemble expectation $\int \sum_{i=1}^{d} P_i T_i(\tau) d\tau$.

III. A Genetic-Based Design Space Exploration

Once the topological structure of the classifier has been fixed as suggested in Section II in terms of master module configuration, subclassifier number and input/output dimensioning, the next design step requires configuration of each subclassifier. In turn, this operation requires, for *each subclassifier*

1) identification of the most appropriate classification model structure (e.g., k-NN, RBF, FF);
2) within each structure, determination of the most suitable classification model family/kernel;
3) configuration/training of each subclassifier.

In other words, we have to identify, dimension and finally train, d subclassifiers to maximize accuracy and trade it off with complexity.

By configuring Φ_{MC} we have to guarantee that the accuracy performance of the obtained classification core is acceptable for the application, i.e., that it is below a tolerated accuracy loss. Such accuracy threshold depends on the intrinsic nature of the classification problem and can be estimated through the best classifier obtained on the available data set (i.e., Φ_M). In other terms we require the multiplexed classifier to be designed to provide acceptable accuracy. Moreover, the designer might tolerate an accuracy loss to gain margin in trading-off accuracy and complexity.

The optimal design problem of a multiplexed classifier can be cast in the optimization problem

$$\begin{cases} \overline{\Phi}_{MC} = \min_{\Phi_{MC} \in \Theta} C(\Phi_{MC}) \\ A(\Phi_{MC}) \leq A_T \end{cases} \tag{4}$$

where Φ_{MC} is the multiplexed classifier belonging to the classification design space Θ, $A(\Phi_{MC})$ is the accuracy performance of Φ_{MC}, A_T represents the maximum accuracy loss tolerated for the application and $\overline{\Phi}_{MC}$ is the optimal multiplexed classifier. Accuracy can be computed by using validation techniques such as cross-validation, N-fold cross-validation, LOO [21], [28]. In the following, we consider cross-validation techniques for validating classifier Φ_{MC} applied to a mean squared error loss function.

Solution to problem (4) is particularly complex due to its nonlinear nature. A simplistic approach based on a greedy philosophy would be surely not appropriate since a large number of unfeasible solutions with respect to accuracy would be generated with an obvious cost in the design phase. We can then follow a penalising approach envisaging a penalization term to accuracy which accounts for complexity (e.g., see [29]–[31]). A different approach would involve Pareto optimal borders. A simple figure of merit trading off complexity and accuracy is

$$J_{M,MC} = \min_{\Phi_{MC} \in \Theta} \left(\gamma \frac{C(\Phi_{MC})}{C(\Phi_M)} + \frac{A(\Phi_{MC})}{A(\Phi_M)} \right) \tag{5}$$

where both complexity and accuracy of the Φ_{MC} classifiers are normalized with respect to the reference best performing classifier. Normalization allows the designer for easily comparing relative behaviors of competing terms. $\gamma \geq 0$ is a penalising constant fixed by the designer, weighting the accuracy and complexity contributions; a small γ implies that the accuracy constraint is more relevant than the complexity one (for $\gamma = 0$ the optimization is driven only by accuracy).

Here we opted for the traditional genetic algorithms approach suggested in [32], [33] for its immediate implementation and blind approach. Obviously, the designer can select his/her favorite figure of merit/optimization techniques.

Each chromosome codes the structure of a given multiplexed classifier Φ_{MC}. More in detail, the chromosome contains information related to *model type* and *topological complexity* for each of the d subclassifiers composing the classifier Φ_{MC} (the chromosome is composed of $2d$ genes): The weights/parameters of the classifier are not represented in the chromosome which only contains topological and structural information. In particular, the first gene codes the model structure and assumes values 0, 1, 2, 3 associated with k-NN, FF, RBF, and linear classifiers, respectively (for instance, a 1 in the gene of the ith subclassifier require the subclassifier to be implemented with a feedforward neural network). Of course, we could decide to enlarge the gene alleles by adding other classifier structures such as Support Vector Machines [34], [35] and Bayesian classifiers [36]. The second gene codes the value k in k-NN models and the number of hidden units in FF and RBF, respectively (for instance, if the first gene is 1 and the second gene contains 5 it means that we have a FF neural network with five hidden units). The RBF variance was kept fixed but it can become a free parameter to be tuned by coding it into a gene to be added to the chromosome, as any other parameter we wish to automatically identify.

Genetic evolution, by optimising the fitness function $J_{M,MC}$, will push chromosomes undergo reproduction and relevant building blocks related to family and structure of the subclassifiers will be transmitted to offsprings. For each chromosome (multiplexed configuration Φ_{MC}) we have to evaluate the fitness function and, as such we have to training/configure each subclassifier depending on the information coded in the genes. In particular, we adopted a Levenberg-Marquardt method with early stopping based on the test set to keep under control overfitting phenomena [11] for neural networks and the standard configuration procedure [28] for k-NN. Again, the designer can consider different approaches without affecting the methodology. It should be noted that the chromosome characterizes the classifier in terms of its topology (and, hence, complexity) while accuracy performance can be estimated only when each model has been configured, i.e., we have classifier Φ_{MC}. Once the parameters of the subclassifiers have been identified a cross-compiler automatically transforms the multiplexed classifier code in a C-code to be given to the WATTCH simulator for estimating the classifier complexity. After this step, the figure of merit (3.2) can be computed and the genetic engine evolve.

IV. Experimental Results

In this section, we apply the design methodology to six applications. The first three $I1$–$I3$ refer to nontrivial industrial applications. Dataset $I4$, $I5$, and $I6$ are benchmarks taken from

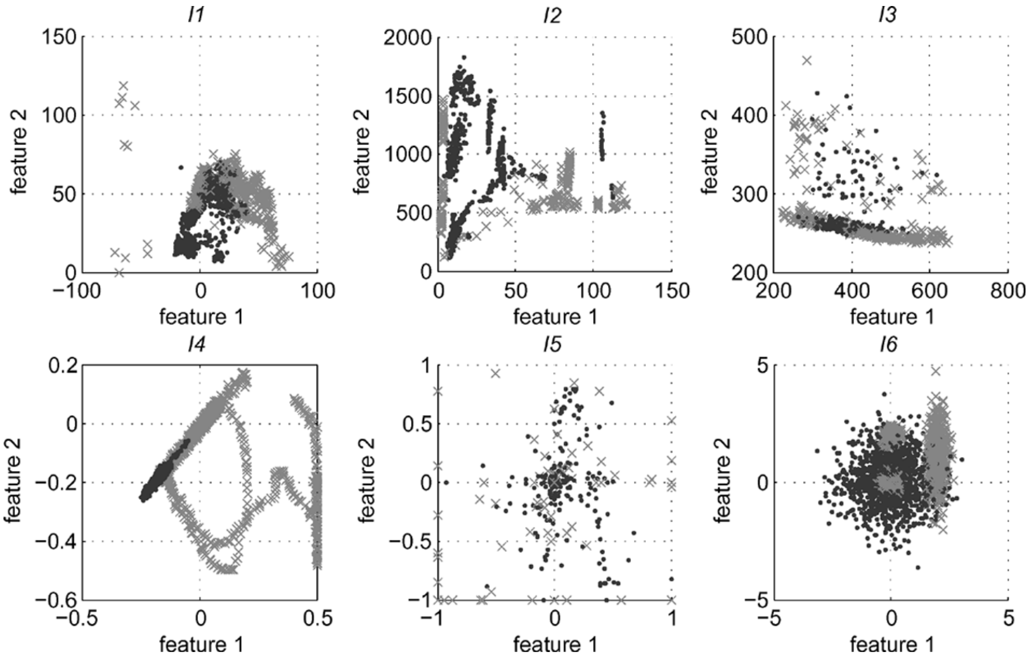

Fig. 4. Six application benchmarks. The plots represent the distribution of the good (.) and bad (x) samples projected over the two most relevant features subspace.

DAMADICS Research Training Network [37], UCI dataset repository [38], and the ELENA project [38], [39], respectively.

More specifically, the first industrial application—$I1$—refers to a quality analysis process for the stainless steel laser cutting industry [17]: The goal is to judge, directly during the operational phase, the local quality of the cut. We discovered, by applying the feature selection procedure suggested in Fig. 3, that 6 features are relevant to the classification problem. The identified features are the cutting speed, the pressure of the shielding gas and four features associated with the camera inspection of sparks produced during the cutting: The spark jet presence, the angle of the sparks core with respect to the normal, its wideness angle and the angle containing the whole sparks jet. A view of the good (dot points)–no good (x points) classification space is given in Fig. 4 with respect to the two most relevant features (we will keep the same graphical notation for the other experiments).

The second industrial application—$I2$—is related to a 3d-laser scanner for railroad tracks profile analysis [40]. The detection system consists of a laser source, whose beam is collimated by an optic lens into a light plane, and two CCD cameras for complete observation of the track. Two digital processing systems (one per camera) extract the track profile from the retrieved images. A classification core is needed to identify, within each column of the images, the presence of a laser reflection. Seven features have been extracted: Minimum and maximum intensity of the column, their difference, the standard deviation of the intensity (*std*), the maximum value of the convolution of the column intensity with a reference Gaussian pattern (*conv*), the difference $std - conv$, and the energy E of the derivate of the column intensity.

The third industrial application—$I3$—is related to a quality analysis for the laser spot welding in the electronics manufacturing industry [18]. We identified that the relevant features to the classification problem (good/no-good weld) are the laser pulse energy, time of the first significant minimum of the back-reflected laser power signal, the time of turnpoint for the temperature sensor and the plume delay.

The fourth dataset—$I4$—was generated from a sugar factory process data [37] to design an actuator diagnosis system in industrial control. The dataset contains 32-dimensional samples measured from sensors applied to of the sugar plant. We considered two features extracted from the samples: The residuals of the rod displacement and the juice flow of the first plant actuator during 10 000 seconds of functioning (30 October–17 November 2001). We grouped the classification patterns into "fault" and "no fault" situations.

The fifth dataset—$I5$—refers to the Ionosphere problem; it contains 34 features extracted from radar signals coming from 16 high-frequency antennas at the radar station of Goose Bay, Labrador, USA. The experiment is subdivided into two classes: "Good" radar returns (those signals showing evidence of structure in the ionosphere) and "Bad" returns (those signals that passed through the ionosphere without reflection).

Dataset—$I6$—refers to the Clouds dataset from the Elena Project. Data are synthetic; class 0 is the sum of three Gaussian distributions, class 1 is ruled by a single Gaussian distribution. There is an important overlap between the two classes leading to a theoretical Bayesian error equal to 9.66%.

After having applied the first three steps of the methodology we identify the topological structure and the P_is of the subclassifiers as given in Table I. The table contains information characterising the master module as well as the subclassifiers composing the multiplexed classifier. We observe that, depending on the particular nature of the application, the number of subclassifiers, their activation probability P_i, the number of features and model nature, can be significantly different. The table also contains, for sake of completeness, the complexity C_i of each subclassifier and the subclassifier type, information derived from

TABLE I

Configuration of the Φ_{MC}s After Having Applied Steps 1–4 of the Methodology; *Type*=Classifier Type (FF=Feedforward, DEG=Degenerated, LIN=Linear), P_i=the Activation Probability of the i-th Subclassifier, *Features*=the Envisioned Number of Features, C_i and $C\mathrm{Master}$ Represents the Complexity, Measured, in Cycles by the WATTCH Profiler of the Subclassifier and the Master Module, Respectively

Datasets		*I1*	*I2*	*I3*	*I4*	*I5*	*I6*
Master module	C_{Master}:	636.70	452.60	456.70	276.70	1802.60	412.70
	Features #:	6	7	4	2	34	2
M1	Type:	DEG	FF [2 1]	FF [6 1]	DEG	LIN	FF [2 1]
	P_1:	0.25	0.14	0.37	0.14	0.54	0.15
	Features #:	0	7	3	0	34	1
	C_1:	1	560.54	1107.79	1.00	562.25	380.34
M2	Type:	FF [4 1]	FF [5 1]	FF [5 1]	FF [2 1]	LIN	FF [2 1]
	P_2:	0.29	0.86	0.18	0.32	0.46	0.24
	Features #:	5	5	4	2	5	2
	C_2:	946.65	1131.69	1036.72	400.52	98.25	400.52
M3	Type:	FF [6 1]		FF [4 1]	FF [2 1]		FF [2 1]
	P3:	0.46		0.45	0.54		0.25
	Features #:	4		4	2		2
	C_3:	1131.69		818.66	400.52		400.52
M4	Type:						DEG
	P_4:						0.16
	Features #:						0
	C_4:						1
M5	Type:						FF [2 1]
	P_5:						0.2
	Features #:						2
	C_5:						400.52

step 4 of the methodology. For instance, if we consider application $I2$ we have that the master module receives seven features and is characterized by a 452.60 (code execution cycles) complexity as profiled by WATTCH; the multiplexed classifier contains two submodules M1 and M2. M1 is a feedforward neural network receiving seven features, it has two hidden units and a single output unit (i.e., FF [2 1]) while M2 is a feedforward neural network with five hidden units and receives five input features. M1 has activation probability 0.14 and profiled complexity 560.54 while M2 is characterized by an activation probability of 0.86 and a complexity of 1131.69 cycles.

We study at this point, the $T_i(\tau)$ curves for the experiments to discover whether locality properties at application level hold or not and, as such, if we can expect a significant gain in complexity and power consumption by resorting to multiplexed classifiers. Results are given in Fig. 5 where each subfigure contains the $T_i(\tau)$ curves associated with all subclassifiers composing a configured multiplexed classifier. As reference line we considered the $d^{-\tau}$ curve (plotted as a continuous line) which represents the $T_i(\tau)$ curve associated with an application with d subclassifiers whose inputs are independent and identically distributed subject to a uniform distribution. Under such hypotheses there is no time-dependency and temporal locality is null. The more a $T_i(\tau)$ curve is above the reference curve the more the subclassifier possesses the temporal locality property.

We appreciate the fact that industrial applications $I1$-$I4$ show a good temporal locality, as we could have expected from the continuous nature of the application while curves associated with applications $I5$ and $I6$ do not possess temporal locality (as expected). For instance, module $\Phi_{MC,2}$ of application $I2$ shows a high activation probability from Table I which means that the subclassifier ($P_2 = 0.86$) is more active than its companion ($P_1 = 0.14$): In a Least Recently Used cache block substitution policy the module will stay in the cache with high probability (we also have to remind that subclassifiers are in general smaller than the monolithic one). In addition, $\Phi_{MC,2}$ possesses an extremely good temporal locality profile from Fig. 5: The probability that $\Phi_{MC,2}$ will be active for all next 20 input patterns is above 0.8, hence, leading the whole classifier complexity and also impacting on cache miss reduction.

We can finally consider step 4 of the methodology, which requires the creation of the Φ_{MC} classifiers. The reference classifier Φ_M required in (5) has been selected among a set of monolithic classifiers. The set contained feedforward neural networks with a number of hidden neurons in the range $[1, 2, \ldots, 30]$ trained ten times with the Levenberg-Marquardt method [41] and Bayesian regularization [42], [43] and k-Nearest Neighbor classifier, k in the $[1, 3, 5, \ldots, 15]$ range.

The most accurate classifier present in the generated classifier set became Φ_M. Accuracy functions $A(\Phi_M)$ and $A(\Phi_M)$ are based on MSE cross-validation. The population required by GAs was set to of 20 randomly initialized Φ_{MC}; the standard genetic algorithms procedure evolved for 100 generations.

The whole automatic design system has been implemented in Matlab by exploiting the available Neural Network Toolbox and PRTOOL [44]. For genetic optimization we integrated the GATOOL toolbox [45] in our design system.

Table II presents a comparison among different classifier families with respect to accuracy (*A*, estimated with cross-validation represents the percentage classifier error), complexity (*C*, in code execution cycles, as estimated by WATTCH) and power consumption (*W*, in power units, as estimated by WATTCH). The first two classifiers belong to the multiple classifier class as suggested in [44] (of course, many other combination could have been selected, e.g., see [1], [46]). Here, we considered two groups of classifiers, each of which composed of three modules

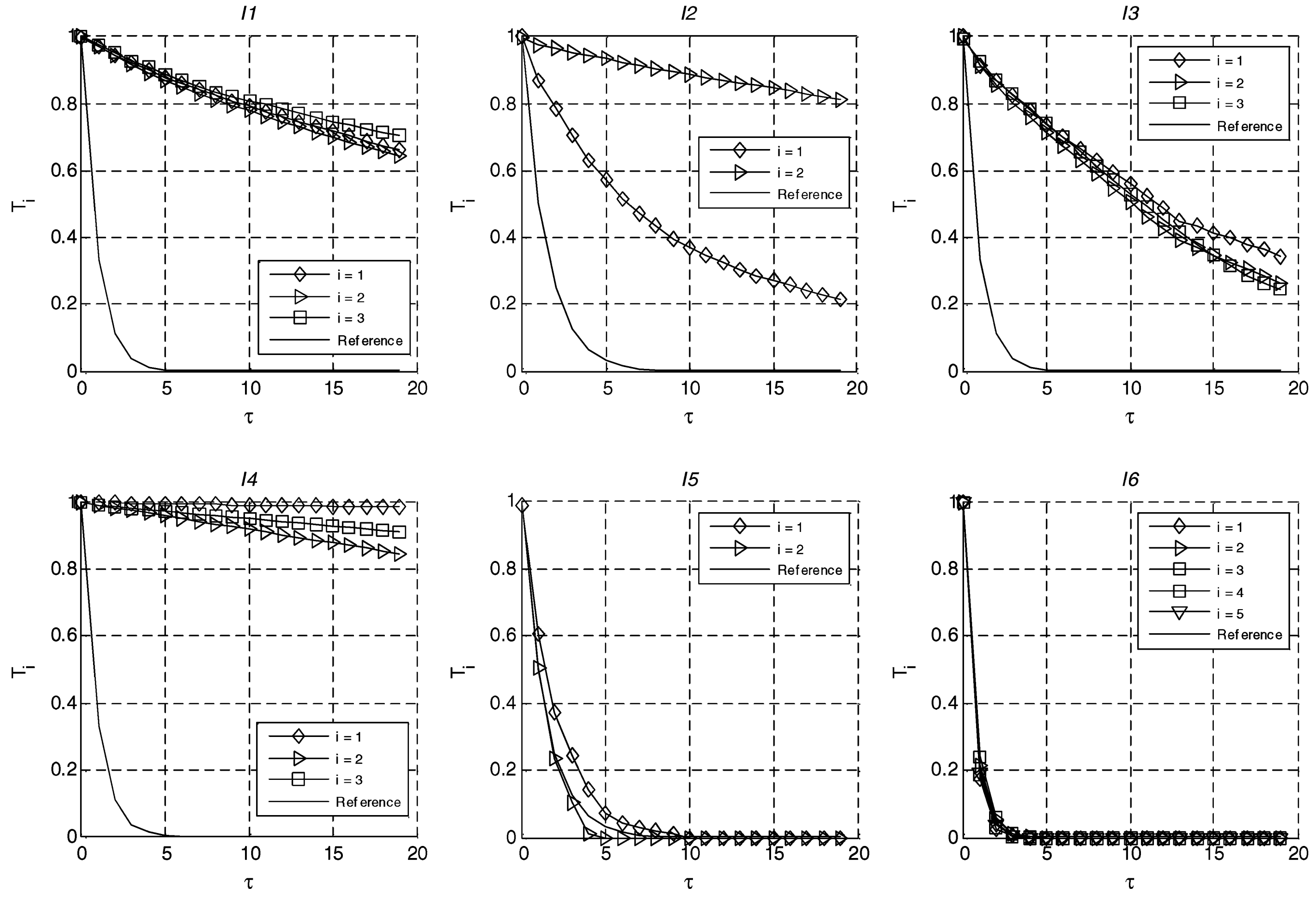

Fig. 5. $T_i(\tau)$ curves.

TABLE II
COMPARATIVE RESULTS (A=ACCURACY, C=COMPLEXITY MEASURED IN CLOCK CYCLES, W=POWER ENERGY UNITS AS PROVIDED BY WATTCH)

	Best Multiple Cl. Group 1			Best Multiple Cl. Group 2			Best k-NN			Best FFNN			Best Φ_{CM}		
	A	C	W	A	C	W	A	C	W	A	C	W	A	C	W
I1	3.8%	715028	51809610	4.4%	5684	411844	3.5%	709459	51406045	**0.3%**	8338	604132	1.8%	846	61295
I2	1.0%	254464	18437990	2.6%	6144	445174	**0.9%**	248451	18002254	1.0%	2290	165926	1.2%	1741	126171
I3	5.0%	257817	18680812	3.3%	4764	345184	2.3%	253136	18341721	**1.7%**	4359	315830	3.3%	1771	130542
I4	3.7%	3793	274745	3.0%	3483	252260	3.1%	406324	29441512	**3.0%**	3053	221189	3.3%	781	56562
I5	**2.9%**	391987	28402570	5.7%	18564	1345104	5.7%	373986	27098339	5.7%	2377	172202	5.7%	1935	140171
I6	12.5%	206776	14982511	11.6%	3844	278519	15.9%	202983	14707766	**11.1%**	5992	434172	12.0%	923	66850

(our multiplexed classifiers considered in average three subclassifiers). The first group of multiple classifiers contains a k-NN classifier and two feedforward neural networks as suggested in [44] (i.e., 4 and 20 hidden units with linear output neuron models). The second group is similar to the first one but substitutes the k-NN with a linear classifier. By following the methodology delineated in [1] and [44] we combined the classifiers' output with the product, mean, median, maximum, minimum, and voting algorithms and selected the most accurate multiple classifier based on cross-validation.

The third and the fourth classifiers are the best monolithic k-NN and monolithic feedforward we obtained on the datasets by exploring k and the number of hidden units. Finally, Table II presents information associated with the best multiplexed classifier identified with the proposed methodology. In bold italic font, we show the most accurate classifier for each dataset which, in the following becomes the reference Φ_M with accuracy $A(\Phi_M)$.

As we could have expected our designed multiplexed classifier behaves slightly worse that the best one. In fact, through minimization of the figure of merit defined in (5), we aim at trading off accuracy with complexity (i.e., we accept a slight loss in accuracy in favor of an improvement in complexity and power consumption).

Table III presents comparative results; there Φ_M is the best reference classifier identified in Table II and Φ_M the multiplexed one. In column one we have the tolerated loss in performance $\Phi_M - \Phi_{MC}$, while in the following columns we have the estimated gain in code execution cycles, power, and real execution time, respectively.

TABLE III
ACCURACY LOSS, GAIN IN COMPLEXITY (*C*), POWER (*W*) AND EXECUTION TIME (*T*)

Dataset	$A(\Phi_{MC}) - A(\Phi_M)$	$\frac{C(\Phi_M)}{C(\Phi_{MC})}$ Gain (cycles)	$\frac{W(\Phi_M)}{W(\Phi_{MC})}$ Gain (power)	$\frac{T(\Phi_M)}{T(\Phi_{MC})}$ Gain (time)
I1	1.5%	9.86	9.86	10.20
I2	0.3%	142.68	142.68	122.13
I3	1.6%	2.46	2.42	2.48
I4	0.3%	3.91	3.91	3.84
I5	2.8%	202.63	202.62	192.12
I6	0.9%	6.49	6.50	6.01

The execution time was measured on a real processor (INTEL P4 2 GHz with 750 MB RAM mounting the Microsoft WINDOWS Xp operating system) by removing unnecessary operating system processes; measures were averaged over 100 presentations of the whole data set for each benchmark to reduce the impact of unremovable operating system processes. We observe that the estimate in complexity provided by WATTCH well matches with the real gain in execution time. The good accuracy of the estimate is a fundamental result since the genetic optimization evolves on the basis of such information.

Our experiments show that the methodology can produce multiplexed classifiers whose accuracy is comparable to the ones designed without the methodology but with a significant reduction in complexity and power consumption. In some cases the complexity gain, which expresses how many times the reference classifier is more complex than the multiplexed one, is very high. We observed that the gain in complexity reduces (see $I4$) when, due to the nature of the application, the reference classifier itself is simple: Partitioning in subclassifiers provide little additional value.

V. CONCLUSION

This paper presents a methodology for designing embedded classifiers which is particularly effective in those applications possessing locality properties, i.e., strong temporal and spatial relationships for the input patterns to be given to the classifier. A variant of the gated-parallel classification structure has been found particularly suitable since it allows for exploiting locality properties, hence, providing classifiers that are less complex and power consuming than those obtainable with a traditional design. Results, applied to a large set of applications, show that the gain in computational complexity and power consumption reduction can be very high depending on the strength of the locality property and the complexity of the optimal classifier. A relevant gain in execution time and energy saving make the proposed solution particularly suitable to embedded systems encompassing a classification core. In particular, by reducing complexity we can reduce the clock frequency yet keeping the same throughput, with an additional gain in power consumption reduction.

REFERENCES

[1] A. K. Jain, R. P. W. Duin, and J. Mao, “Statistical pattern recognition: a review,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 1, pp. 4–37, 2000.

[2] R. P. W. Duin, “A note on comparing classifiers,” *Pattern Recognit. Lett.*, vol. 17, no. 5, pp. 529–536, 1996.

[3] M. H. Hassoun, *Fundamentals of Artificial Neural Networks*. Cambridge, MA: MIT Press, 1995.

[4] C. Alippi and L. Briozzo, “Accuracy vs. precision in digital vlsi architectures for signal processing,” *IEEE Trans. Comput.*, vol. 47, no. 4, pp. 472–477, 1998.

[5] S. Piche, “The selection of weights accuracies for madalines,” *IEEE Trans. Neural Netw.*, vol. 6, no. 2, pp. 432–445, 1995.

[6] A. U. Levin, T. K. Leen, and J. E. Moody, “Fast pruning using principal components,” in *Proc. Adv. Neural Info. Process. Syst.*, vol. 6, 1994, pp. 35–42.

[7] Y. LeCun, J. Denker, S. Solla, R. E. Howard, and L. D. Jackel, “Optimal brain damage,” in *Proc. Adv. Neural Info. Process. Syst. II*, D. S. Touretzky, Ed., 1990.

[8] B. Hassibi and D. G. Stork, “Second order derivatives for network pruning: optimal brain surgeon,” in *Proc. Adv. Neural Info. Process. Syst.*, vol. 5, 1993, pp. 164–171.

[9] I. Guyon, V. Vapnik, B. Boser, L. Bottou, and S. A. Solla, “Structural risk minimization for character recognition,” in *Proc. Adv. Neural Info. Process. Syst.*, 1992, pp. 471–479.

[10] C. Alippi and V. Piuri, “Topological minimization of multi-layered feed-forward neural networks by spectral decomposition,” in *Proc. IEEE Int. Joint Conf. Neural Netw.*, Nov. 1992, pp. 805–810.

[11] B. D. Rypley, *Pattern Recognition and Neural Networks*. Cambridge, U.K.: Cambridge Univ. Press, 1996.

[12] M. Jordan and R. Jacobs, “Hierarchical mixtures of experts and the EM algorithm,” *Neural Comput.*, vol. 6, pp. 181–214, 1994.

[13] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[14] I. T. Jolliffe, *Principal Component Analysis*. New York: Springer-Verlag, 1986.

[15] B. Schölkopf, A. J. Smola, and K.-R. Müller, “Nonlinear component analysis as a kernel eigenvalue problem,” *Neural Comput.*, vol. 10, no. 5, pp. 1299–1319, 1998.

[16] A. Blum and P. Langley, “Selection of relevant features and examples in machine learning,” *Artif. Intell.*, vol. 97, no. 1–2, pp. 245–271, 1997.

[17] C. Alippi, V. Bono, V. Piuri, and F. Scotti, “Toward real-time quality analysis measurement of metal laser cutting,” in *Proc. IEEE Int. Symp. Virtual and Intell. Measure. Syst.*, May 2002, pp. 34–44.

[18] C. Alippi, P. Braione, V. Piuri, and F. Scotti, “A methodological approach to multisensor classification for innovative laser material processing units,” in *Proc. IEEE Instrum. Measure. Technol. Conf.*, Budapest, Hungary, 2001, pp. 1762–1767.

[19] P. H. R. Duda and D. Stork, *Pattern Classification*, 2nd ed. New York: Wiley, 2001.

[20] P. A. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach*. London, U.K.: Prentice-Hall Int., 1982.

[21] K. Fukunaga, *Introduction to Statistical Pattern Recognition*. New York: Academic, 1972.

[22] K. Fukunaga and R. R. Hayes, “Effects of sample size in classifier design,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, no. 8, pp. 873–885, 1989.

[23] F. Aurenhammer, “Voronoi diagrams—a survey of a fundamental geometric data structure,” *ACM Comput. Surv.*, vol. 23, no. 3, pp. 345–405, 1991.

[24] A. Lipnickas, “Classifiers fusion with data dependent aggregation schemes,” in *Proc. Int. Conf. Info. Netw., Syst. Technol.*, Minsk, Belarus, 2001, pp. 147–153.

[25] L. I. Kuncheva, “Cluster-and-selection method for classifier combination,” in *Proc. Int. Conf. Knowledge-Based Intell. Eng. Syst. Allied Technol.*, Brighton, U.K., 2000, pp. 185–188.

[26] D. Brooks, V. Tiwari, and M. Martonosi, “WATTCH: a framework for architectural-level power analysis and optimizations,” in *Proc. Int. Symp. Comput. Architec.*, 2000, pp. 83–94.

[27] T. Austin, E. Larson, and D. Ernst, “Simplescalar: an infrastructure for computer system modeling,” *Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[28] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.

[29] P. J. Edwards and A. F. Murray, “Toward optimally distributed computation,” *Neural Comput.*, vol. 10, no. 4, pp. 987–1005, 1998.

[30] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Proc. Adv. Neural Info. Process.*, 1992, pp. 950–957.

[31] G. Seber and C. Wild, *Nonlinear Regression*. New York: Wiley, 1989.

[32] L. I. Kuncheva and L. C. Jain, "Designing classifier fusion systems by genetic algorithms," *IEEE Energy Conv.*, vol. 4, no. 4, p. 327, Nov. 2000.

[33] P. Lanzi and R. L. Riolo, "Advances in evolutionary computing: theory and applications," in *Recent Trends in Learning Classifier Systems Research*, ser. Natural Computing. New York: Springer-Verlag, 2003.

[34] K.-R. Mller, S. Mika, G. Rtsch, K. Tsuda, and B. Schlkopf, "An introduction to kernel-based learning algorithms," *IEEE Trans. Neural Netw.*, vol. 12, no. 2, pp. 181–201, 2001.

[35] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Min. Knowl. Discov.*, vol. 2, no. 2, pp. 121–167, 1998.

[36] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine Learning*, vol. 29, no. 2–3, pp. 131–163, 1997.

[37] EC FP5 Research Training Network DAMADICS. Development and Application of Methods for Actuator Diagnosis in Industrial Control Systems. [Online]http://www.eng.hull.ac.uk/research/control/damadics1.htm

[38] C. Blake and C. Merz. UCI Machine Learning Databases Repository, University of California-Irvine, Department of Information and Computer Science. [Online]ftp://ftp.ics.edu/pub/machinelearningdatabases

[39] C. A. Cruz, D. V. Cappel, G. Guerin-Dugue, and C. Jutten. (1995) Deliverable r3-b1-p Task b: Databases, Elena-Nervesii Enhanced Learning for Evolutive Neural Architecture. [Online]Tech. Rep. ESPRIT-Basic Research Project Number 6891

[40] C. Alippi, E. Casagrande, V. Piuri, and F. Scotti, "Composite real-time image processing for railways track profile measurement," *IEEE Trans. Instrum. Meas.*, vol. 49, pp. 559–564, Jun. 2000.

[41] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Oxford Univ. Press, 1996.

[42] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and neural networks architectures," *Neural Comput.*, vol. 7, no. 2, pp. 219–269, 1995.

[43] M. Hintz-Madsen, L. K. Hansen, J. Larsen, M. W. Pedersen, and M. Larsen, "Neural classifier construction using regularization, pruning and test error estimation," *Neural Netw.*, vol. 11, no. 9, pp. 1659–1670, 1998.

[44] F. van der Heiden, R. Duin, D. de Ridder, and D. Tax, *Classification, Parameter Estimation, State Estimation: An Engineering Approach Using MatLab*. New York: Wiley, 2004.

[45] A. Chipperfield, P. Fleming, and C. Fonseca, "Genetic algorithm tools for control systems engineering," in *Proc. Adapt. Comput. Eng. Des. Contr.*, 2000, pp. 950–957.

[46] R. Liu and B. Yuan, "Multiple classifiers combination by clustering and selection," *Info. Fusion*, vol. 2, pp. 163–168, 2001.

Cesare Alippi (SM'94–F'05) received the Dr.Ing. degree in electronic engineering (*summa cum laude*) in 1990 and the Ph.D. degree in computer engineering in 1995, both from the Politecnico di Milano, Milan, Italy. He has completed research work in computer sciences at University College, London, U.K., and the Massachusetts Institute of Technology, Cambridge.

Currently, he is a Full Professor in Information Processing Systems at the Politecnico di Milano. His research interests include application-level analysis and synthesis methodologies for embedded systems, neural networks, and wireless sensor networks. His research results have been published in more that 120 technical papers in international journals and conference proceedings.

Fabio Scotti (M'03) received the Ing. degree in electronic engineering in 1998 and the Ph.D. degree in computer engineering in 2003 from the Politecnico di Milano, Milan, Italy.

Since 2003, he has been an Assistant Professor with the Department of Information Technologies, University of Milan. His research interests include high-level system design, signal and image processing, computational intelligence algorithms, and their applications in the industrial field. His current research focuses on design methodologies and algorithms for multimodal biometric systems.