Novel Cascade FPGA Accelerator for Support Vector Machines Classification

Markos Papadonikolakis and Christos-Savvas Bouganis, *Member, IEEE*

***Abstract*— Support vector machines (SVMs) are a powerful machine learning tool, providing state-of-the-art accuracy to many classification problems. However, SVM classification is a computationally complex task, suffering from linear dependencies on the number of the support vectors and the problem's dimensionality. This paper presents a fully scalable field programmable gate array (FPGA) architecture for the acceleration of SVM classification, which exploits the device heterogeneity and the dynamic range diversities among the dataset attributes. An adaptive and fully-customized processing unit is proposed, which utilizes the available heterogeneous resources of a modern FPGA device in efficient way with respect to the problem's characteristics. The implementation results demonstrate the efficiency of the heterogeneous architecture, presenting a speed-up factor of 2–3 orders of magnitude, compared to the CPU implementation. The proposed architecture outperforms other proposed FPGA and graphic processor unit approaches by more than seven times. Furthermore, based on the special properties of the heterogeneous architecture, this paper introduces the first FPGA-oriented cascade SVM classifier scheme, which exploits the FPGA reconfigurability and intensifies the custom-arithmetic properties of the heterogeneous architecture. The results show that the proposed cascade scheme is able to increase the heterogeneous classifier throughput even further, without introducing any penalty on the resource utilization.**

***Index Terms*— Cascade classifier, classification, field programmable gate array (FPGA), parallel processing, support vector machines (SVMs).**

I. Introduction

SUPPORT vector machines (SVMs) [1] are one of the most popular supervised learning instances and they are considered as an effective machine learning method, providing good generalization performance for a wide range of regression and classification tasks [2]. The supervised learning methods are comprised of two discreet phases, the training and the classification (or regression, in cases where the system's output is continuous). The SVM training phase is responsible for the identification of these data points, called *support vectors* (SVs), that can best build a separation model for the classes. These vectors are then used to predict the class of any future data point during the classification phase.

Manuscript received November 19, 2010; revised January 29, 2012; accepted April 5, 2012.

The authors are with the Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2AZ, U.K. (e-mail: markos.papadonikolakis07@imperial.ac.uk; christos-savvas.bouganis@imperial.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2012.2196446

The SVM training, based on a database of known training vectors, can be performed online or offline, depending on the application. The SVM classification is mostly performed in newly obtained data. Applications, such as face detection, speech recognition, bioinformatics or geostatistical analysis often require online classification and have real-time constraints. However, the SVM classification is a computationally expensive task, linearly dependent on the classification load, the SVs population, and the problem's dimensionality. In cases where large-scale problems are targeted or a high classification throughput must be sustained, the classification task becomes very time consuming and urgent needs for acceleration arise.

Depending on the targeted problem, the dataset can be characterized as *homogeneous* or *heterogeneous*. Homogeneous datasets are often met in imaging, like face detection or recognition [3]. In homogeneous datasets, the precision requirements among the dataset features are the same. For example, MNIST [4] consists of 28×28 eight-bit images per training sample and it is a homogeneous problem, since all 784 attributes of its samples require an eight-bit representation. Nevertheless, other real-world datasets present significant diversities among the dynamic ranges of their dimensions. The attributes of these heterogeneous datasets can be continuous, indexing, categorical or boolean. For instance, the attributes of Adult [5] dataset include, among others, a person's age, sex and marital status, these attributes can be characterized as continuous, boolean, and categorical, respectively, and different precision requirements arise between them. One would only need one bit to represent a boolean attribute, whereas the marital status, according to the dataset specifications, falls into seven distinct categories and a three-bit representation is sufficient.

The performance of the SVMs can be maximized by customizing the use of the available computing resources to take into account the nature of the input data. This is a strong motivation for targeting SVM classifiers on computing devices, which can exploit the potential of custom precision arithmetic, like FPGAs. FPGAs are semiconductor devices, which contain programmable logic elements and a hierarchy of programmable interconnects. In nowadays, FPGAs contain, in addition, coarse grain components, such as memory blocks and embedded multipliers or digital signal processing blocks (DSPs). The implantation of hard logic, such as multipliers onto the programmable fabric has enabled the FPGA devices to boost their performance efficiency. Modern FPGA devices offer a vast amount of DSP blocks and a hierarchy of different memory sizes, providing high level

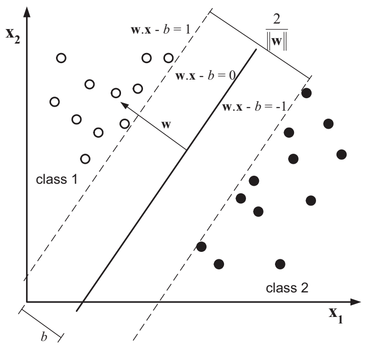

x_2
x_1
$\mathbf{w}.\mathbf{x} - b = 1$
$\mathbf{w}.\mathbf{x} - b = 0$
$\mathbf{w}.\mathbf{x} - b = -1$
$\frac{2}{\|\mathbf{w}\|}$
$\mathbf{w}$
class 1
class 2
b

Fig. 1. SVM separating hyperplane.

of flexibility and large amount of parallel computational power. Moreover, their portability allows them to be used as ad-hoc solutions, in cases where the application suffers from bandwidth limitations [6]. The FPGA reconfigurability offers a significant advantage against application-specific standard products and application-specific integrated circuits, when targeting different classification problems which may vary in size, dimensionality, and dynamic range constraints. Additionally, modern FPGA devices are able to offer equal or superior performance at a lower power cost than general purpose graphic processing units.

The linear dependencies of the classification task complexity indicate that the execution time can be improved by reducing the number of SVs used in the decision function. Taking this idea a step further, works like [7]–[9] proposed the usage of cascade schemes, in order to speed-up the SVM classification task. These architectures combine classifiers in a cascading fashion, where a succeeding classifier builds on the output of the previous one in order to produce a classification decision.

This paper proposes a new FPGA-oriented scheme for the acceleration of the SVM classification. The novelty of this scheme lies in the exploitation of the FPGA custom-arithmetic potential, in order to utilize the classifiers of the cascade chain under different arithmetic precision. First, a fully scalable heterogeneous FPGA architecture is proposed, whose objective is the exploitation of the precision requirements of the problem's attributes, in order to use the available resources of a modern FPGA device in the most efficient manner. The maximum parallelization factor for the SVM training processor is achieved by maintaining the device's ratio between DSPs and logic resources. The proposed FPGA architecture results demonstrate a speed-up factor of 7 and 8, when compared to previous FPGA and graphic processing unit (GPU) architectures.

Taking this a step further, the proposed cascade classifier enhances the characteristics of the heterogeneous classifier and exploits the relationship between the precision and the resource utilization in a more efficient manner. Moreover, the proposed cascade architecture employs the FPGAs' reconfigurability and expands the possible design space of the architecture, in order to target problems which do not allow for a fully-unrolled cascade scheme, due to their resource constraints. The implementation results show that a significant gain on throughput or resource usage can be obtained, in cases where the available resource budget does not allow for a full-precision, fully-unrolled implementation, or where extra throughput is desired.

The main contributions of this paper are as follows.

1) The presentation of a novel FPGA architecture for the SVM classification, which fully exploits the parallel processing power of the FPGA heterogeneous resources, outperforms previous FPGA and GPU accelerators, and offers scalability and adaptivity to the targeted classification problem's nature with respect to the available resource constraints.
2) The introduction of the first cascade SVM classifier in the literature that exploits the custom-arithmetic potential of FPGA devices, in order to boost the SVM classification time even further.
3) The employment of FPGA reconfigurability to the cascade classifier, in order to expand the possible design space of the cascade scheme and increase its potential to target the large-scale problems for which the resource budget does not allow for a full cascade implementation.

The rest of this paper is organized as follows. Section II gives the SVM theoretical background. In Section III, the heterogeneous FPGA implementation for the SVM classifier is described, while the proposed cascade classifier architecture is presented in Section IV. Section V provides the implementation results, and this paper concludes in Section VI.

II. SVMs

A. Training

The SVM training builds a model that is able to distinguish the belonging class of any future data based on the SVs obtained by the training dataset. On a two-class classification problem, the SVMs objective is the construction of a separating hyperplane $\mathbf{w} \cdot \mathbf{x} - b = 0$ to attain maximum separation between the classes, as shown in Fig. 1. The classes' hyperplanes are parallel to the separating one, lying on each of its sides. The Euclidean distance between the two hyperplanes is $2/||\mathbf{w}||$, thus the objective of SVMs is to maximize the distance between the classes' hyperplanes or, in other words, to minimize $\|\mathbf{w}\|$

$$\min \frac{1}{2}\|\mathbf{w}\|^2, \quad \text{s.t. } y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1, 1 \leq i \leq N \tag{1}$$

where $(\cdot)$ denotes an inner product, $\mathbf{x}_i$ is the training data, label y_i denotes the belonging class of datum $\mathbf{x}_i$ and takes the values -1, 1, $\mathbf{w}$ is the perpendicular vector to the hyperplane direction, b is the offset to the origin, and N is the training set size. The SVM training phase focuses on the identification of the SVs, which are the training samples that lie closest to the hyperplane and determine its direction.

Solving the SVM training problem, by using quadrating programming (QP) techniques, is a computationally expensive task, especially for large high-dimensional datasets. Hence, many algorithms have been proposed to decompose the QP problem into smaller ones, like the sequential minimum optimization [10], SVM$^{\text{LIGHT}}$ [11] or SVM$^{\text{PERF}}$ [12]. Some other works [13], [14] approach the problem from a geometric point of view. The most important aspect of these solutions is that, for each iteration, the dot-products between a datum and all the other dataset points need to be computed.

In many real-world classification problems, it is often not feasible to linearly separate the data in the original space. SVM's characteristic to adopt the more complex or high-featured spaces is trivial for their wide applicability. Whenever the training data are not linearly separable in the input space, the input space is mapped to a higher-dimensional one, where a linear separation may be feasible. Nevertheless, the explicit construction of this higher-dimensional space mapping for all data points is a computational expensive task, especially in large-scale problems, or even not feasible, since the dimensionality of the space might be infinite. SVMs can avoid this by employing kernel functions $K(\mathbf{x}_i, \mathbf{x}_j)$ to replace the inner products in the optimization problem (1)

$$\min \tfrac{1}{2}\|\mathbf{w}\|_2^2, \quad \text{s.t. } y_i(K(\mathbf{w}, \mathbf{x}_i) - b) \geq 1, \ 1 \leq i \leq N. \quad (2)$$

Out of many possible kernel functions, of special interest are those which satisfy Mercer's condition [15] and can be expressed as an inner product in the high-dimensional space. By applying the kernel, there is no need to explicitly map the data to the higher-dimensional space [16].

B. Classification

Unlike the various algorithmic solvers proposed for the SVM training problem, the SVM classification function is universal and straightforward. On the classification phase, any new datum $\mathbf{x}$ is classified according to the output of the decision function

$$F(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{|N_{\mathrm{SV}}|} y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b\right) \quad (3)$$

where $|N_{\mathrm{SV}}|$ is the cardinality of the set of the SVs identified in the training phase. It is easily derived that the computational time of the nonlinear SVM classification task is linearly dependent on the size of the classification dataset, on the SV population N_{SV} and on the problem's dimensionality. For linear SVMs, the kernel $K(\mathbf{x}_i, \mathbf{x})$ is replaced by an inner product and, taking advantage of its distributivity, the classification complexity does not depend on N_{SV}. For nonlinear SVMs, however, the decision function F requires a massive number of matrix–vector operations for large datasets. However, matrix–vector computations offer significant parallelization potential, which can be exploited by the parallel hardware resources of an FPGA device.

III. Hardware Mapping of the SVM Classifier

This section presents the proposed heterogeneous FPGA-based SVM classifier, which underlies the SVM cascade classifier and is the foundation work for the cascade scheme. First, the related work on hardware-based SVM classifiers is provided, followed by an overview of the heterogeneous FPGA architecture and an in-depth analysis of the *hypertile*, which is the heterogeneous processing element of the FPGA architecture. The last section presents the design flow of the hypertile, and demonstrates how the proposed architecture exploits the dynamic range diversities of any targeted SVM classification problem toward the most resource-efficient and, hence, maximally parallelized classification accelerator.

A. Related Work on Hardware-Mapped SVM Classification

This section overviews some previous FPGA- or GPU-mapped works on the SVM classification. A homogeneous FPGA-based architecture for the SVM training was introduced in [17], and the results can be potentially extended for the acceleration of the SVM classification. Another homogeneous work was presented in [18], where a parallel FPGA co-processor is proposed for the inner product calculations, using the available DSP units of the targeted device. The kernel computations are performed by the host CPU, unlike in [17], where floating-point pipelines are utilized for the kernel functions. The integrated solution in [18] targets a large FPGA device and succeeds in accelerating the SVM classification. Nevertheless, this paper does not exploit the heterogeneity and the fully custom-arithmetic potential of modern FPGA devices and it does not target the precision requirements of the training problem. The multipliers are implemented solely by hard-logic DSP blocks, and the large amount of the FPGA's soft-logic is not efficiently utilized. The work in [19] presents an in-depth analysis of their SVM training architecture on a Xilinx Virtex II device. This paper could potentially be exploited for a classification solution. However, it does not exploit the parallelization potential of modern FPGAs, due to the resource constraints of the targeted device. In [20], a novel implementation based on logarithmic number systems (LNS) is presented. The LNS-based implementation of the SVM kernel is also adopted in [21] in order to produce a hardware-friendly approach. These works focus more on the potential of using LNS for the SVM problem rather than the acceleration of the problem, since the targeted devices are small and only one Multiply-ACcumulate (MAC) unit is used for all the dot-product evaluations. The FPGA architecture proposed in [22] employs a hardware-friendly approach for the kernel evaluations based on CORDIC algorithm, while there is no focus on the maximum utilization of the FPGA logic resources in order to speed-up the SVM classification. The SVM classification is used for a video shit boundary detection in [23]. Only linear SVMs are targeted and the FPGA device is used for the dot-product mapping of the SVM algorithm.

Other works, such as [24] and [25] map the SVM classification problem on the parallel computing resources of a GPU, using NVidia's compute unified device architecture [26] programming environment. Their main differences are related to the chosen floating-point precision for the kernel computations and the usage of the host CPU for the processing of some part of the kernel evaluations, before this is fed to the GPU. Furthermore, the GPU work in [27] targets a geometric interpretation of the SVM training problem [14] based on Gilbert's algorithm [28], while the classification implementation is similar to [24] and [25].

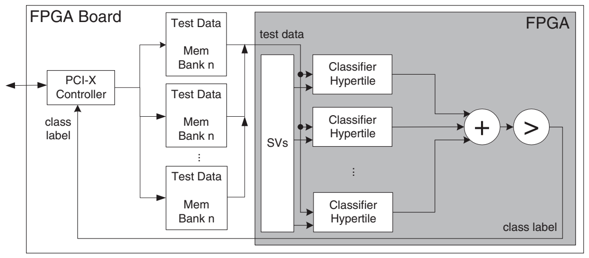


Fig. 2. FPGA architecture of the SVM classifier.

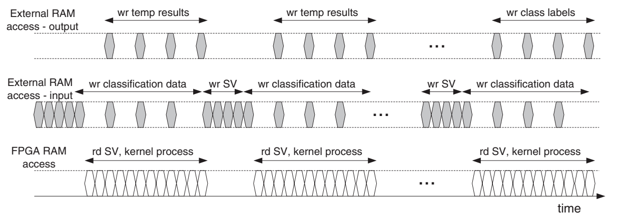


Fig. 3. Memory access timing flow of the proposed FPGA architecture. Read and write RAM operations are abbreviated to "rd" and "wr," respectively. The intermediate results are noted as "temp."

Summarizing, it can be seen that none of the previously presented works focus on the exploration of the vast heterogeneous resources available on an FPGA device with respect to the targeted problem's nature, precision requirements, and characteristics. Most of the works are application-specific and efficient ways to utilize the hardware resources under any targeted problem's constraints are not sufficiently examined. Furthermore, the GPUs are floating-point devices, which mean that resource optimizations based on custom-arithmetic cannot be employed, when targeting a GPU device for the SVM classification. As a conclusion, there is no work proposed so far to the best of authors' knowledge that exploits the heterogeneity in the dataset attributes and investigates a classifier architecture that can fully utilize this property. Even more, there are no available works in the literature to discuss the potential of a cascade, FPGA- or precision-based acceleration of the SVM classification.

B. FPGA Architecture of the SVM Heterogeneous Classifier

The rationale behind the design of the SVM classifier is the exploitation of the parallel computational power offered by the FPGA heterogeneous resources, and the high memory bandwidth offered by the FPGA internal memories in the most efficient way, in order to speed up the decision function (3). The computation of $F(\mathbf{x})$ involves matrix–vector operations, which is highly parallelizable. Therefore, the problem can be segmented into smaller ones and parallel units can be instantiated for the processing of each subproblem.

The proposed FPGA architecture for the SVM classifier is shown in Fig. 2. The SVs are loaded into the internal FPGA memories, while the classification dataset is loaded on the reliability and maintainability symposium (RAMs) of the FPGA board, which serve as First-In, First-Out units between the host and the FPGA. The data points are streamed into the FPGA and fed to each classifier hypertile, which is the processing unit of the architecture. Each hypertile is processing a fragment of the overall classification function. The hypertile outputs the kernel evaluations K, which are then added in parallel. When the sum of (3) is completed, the class of each datum is streamed out by the system. In cases where the SVs cannot fit in the FPGA RAM blocks, the classification is performed into multiple steps, by streaming out the intermediate results and reloading the FPGA RAMs with the next SV subset. Fig. 3 shows the parallel memory accesses of the internal and the external FPGA memories for a multicycle classification task. The FPGA IO-related accesses are colored in gray, while internal FPGA memory accesses are white. The figure shows the processing of only four testing data, for illustration reasons. For large-scale classification problems, the kernel projections are more time-consuming and the idle spaces for the reload of the next SV subset become negligible.

C. Classifier Hypertile

A previous work briefly presented the design flow for a heterogeneous processing unit for the SVM classification [29]. Here, the core idea is presented in more details and exploited for the design of a classifier processing unit, which can be instantiated in the proposed cascade scheme.

The kernel functions embody inner product computations. Thus, the issues regarding the dot-product dynamic range requirements should be first addressed. For a homogeneous dataset with P bits per dimension and dimensionality D, an inner product representation would require $2 \cdot P + \log_2(D)$, fixed-point arithmetic is adequate for a wide range of input characteristics. However, the high-dimensionality of a kernel and its increased dynamic range requires a different approach from the latter case. Floating-point precision is essential for the hardware implementation of kernel functions.

Regarding the need to exploit the different dynamic ranges of a heterogeneous dataset's features, the hardware implementation of an inner product should be efficiently designed. A MAC unit is a good choice when the precision requirements among all dimensions are the same. Nevertheless, if the dataset contains continuous along with categorical or binary attributes, realizing the dot-product as a MAC unit is a waste of resources. Instead, dedicating a custom precision multiplier per attribute significantly reduces the resource usage in dataset cases with large precision diversities. A scheme with parallel multipliers feeding a pipelined adder tree allows for the exploitation of the dynamic range diversities and integrates an adaptive circuit, which uses precisely the resources needed for the data representation.

The architecture of the heterogeneous classifier hypertile is presented in Fig. 4. The data path is split in fixed- and floating-point domains. The internal FPGA memories store an SV subset and feed the parallel multipliers, each of which is dedicated to a single dimension. The features are added according to their precision requirements, in order to minimize the adder tree resource usage. The adder tree produces the inner product—or the norm, in the Gaussian case—for the floating-point kernel processor. The fixed-point inner products are interpreted into IEEE754 single precision

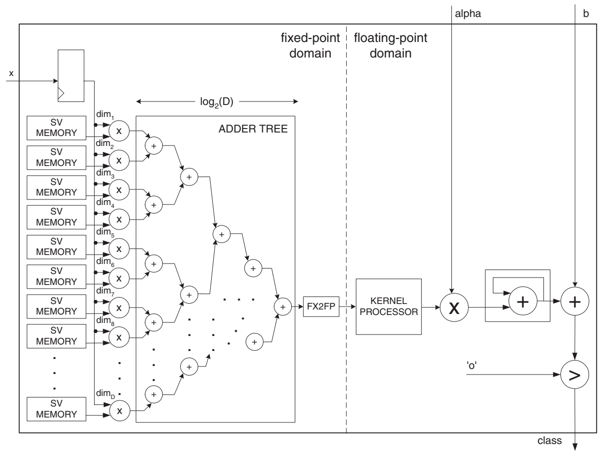


Fig. 4. Hypertile of the heterogeneous SVM classifier.

floating-point format before fed to the kernel. The kernel processor implements one of the three targeted kernel functions, while its output is accumulated to produce the final result of the hypertile.

D. Hypertile Design Flow

The floating-point domain of the hypertile is designed in IEEE754 single floating-point precision, and the hardware design is not dependent on the dataset characteristics. The single floating-point precision is found to be sufficient for a wide range of real-world problems, while it keeps the utilization of the FPGA resources in reasonable levels, compared to the double floating-point implementation. However, the circuits in fixed-point domain are customized according to the precision of each feature. The proposed design flow implements the most resource efficient adder tree, with respect to the targeted problem characteristics. This is achieved by sorting the dataset features in descending precision order and computing the minimum required precision for each node. Furthermore, the design maintains a good ratio between instantiated DSP blocks and lookup tables (LUTs), in order to balance the utilization of the available resources.

The design flow of the heterogeneous classifier hypertile is presented in Fig. 5. After analyzing the targeted input dataset, the tool extracts the categorical and binary attributes and encodes them using the minimum required precision. The dataset is then normalized, a necessary step for the SVMs functionality, so as to prevent a high-offset feature to govern the computations. For more information, the work in [30] presents a useful guide on the SVM preprocessing procedures. The dynamic ranges of continuous features are then computed with a specified error tolerance, allowing for future data diversities. Given the targeted device, the initial ratio between available DSPs and LUTs is computed. Consequently, the algorithm sorts the features in descending dynamic range order to produce the most cost-efficient adders possible and computes the required precision for each node of the fixed-point circuit. Each adder size is dependent on the smallest bit precision between the two operands' decimal parts, since the data are normalized between [0, 1] or [−1, 1]. The adder tree is

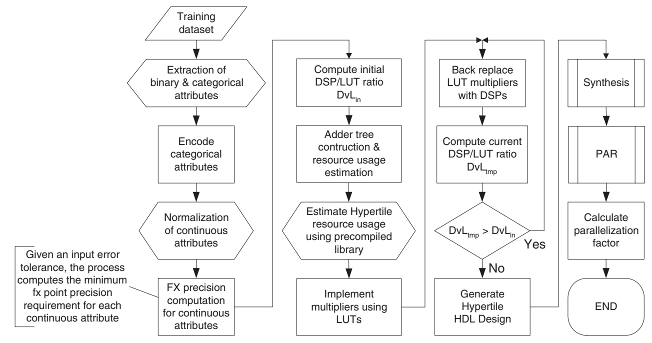

Training dataset
Extraction of binary & categorical attributes
Encode categorical attributes
Normalization of continuous attributes
FX precision computation for continuous attributes
Given an input error tolerance, the process computes the minimum fx point precision requirement for each continuous attribute
Compute initial DSP/LUT ratio DvLin
Adder tree contruction & resource usage estimation
Estimate Hypertile resource usage using precompiled library
Implement multipliers using LUTs
Back replace LUT multipliers with DSPs
Compute current DSP/LUT ratio DvLtmp
DvLtmp > DvLin
Yes
No
Generate Hypertile HDL Design
Synthesis
PAR
Calculate parallelization factor
END

Fig. 5. Heterogeneous architecture, FPGA design flow.

then constructed and the tool estimates the hypertile resource usage, using a precompiled library of the floating-point domain modules. All these processes are completely automated and part of the design flow of the system.

The parallel multipliers are initially instantiated using LUTs and the DSP/LUT ratio is updated, to reflect the resource usage estimates. Starting from the larger precision feature, the LUT-based multipliers are back-replaced with DSP-based ones, until the initial ratio is reached. Thereafter, the hypertile hardware description language design is automatically generated and synthesized.

Summarizing, the proposed flow examines the targeted problem's characteristics and designs the most efficiently utilized processing element, under the resource constraints imposed by the targeted FPGA device. Posterior to the dynamic range analysis and design of the flow, the FPGA device can be reprogrammed and target different classification problems.

IV. Hardware Mapping of the Cascade SVM Classifier

This section presents the proposed FPGA-based SVM cascade scheme. The first section describes the rationale and the main idea of the cascade architecture. It is then followed by two sections, the first one presents the proposed cascade classifier architecture, when the FPGA resources are adequate for its utilization (CC_{FIT}), while the final section demonstrates how the FPGA reconfigurability can be employed for the proposed cascade scheme ($\text{CC}_{\text{RECONF}}$), in order to expand the available design space and address tightly resource-constrained problems that do not allow for a fully unrolled cascade classifier. Table I provides the explanation of the symbols used throughout the rest of this section.

A. FPGA-Based Cascade Scheme

The computational complexity of the SVM classification has driven the research to the proposal of multiple-SVM systems, such as the cascade classifier. This algorithm describes a system of many SVM classifiers in a cascade fashion, where each classifier uses a reduced set vector (RSV) [7] and feeds its output to the next one. The main idea is to bias each classifier in the cascade in a way that one of the binary decisions is very confident, while the other is uncertain and propagates the data

TABLE I
TABLE OF NOTATIONS

Symbol	Meaning
N_X	Set of SVs of classifier X.
LP	Low precision classifier in the cascade chain.
HP	High precision classifier in the cascade chain.
f_X	Decision function of classifier X.
b_X	Offset for decision function of classifier X.
$K_X(\cdot,\cdot)$	Kernel function of classifier X.
C_{cn}, C_{cp}	Classification thresholds.
BC	The FPGA-based heterogeneous classifier, presented in Section III.
CC_{FIT}	Cascade classifier fitting in the FPGA device (Section IV-B).
CC_{RECONF}	Reconfigurable cascade classifier (Section IV-C).
A_X	Resource utilization of classifier/module X.
T_X	Throughput of classifier/module X.
t_X	Classification time of classifier/module X.
lt_X	Support vector loading time of classifier/module X.
pt_X	Processing time of classifier/module X.
t_R	Reconfiguration time of the FPGA device.

point to the next cascade level. The research focuses on the distribution of the reduced SV sets among the classifiers in the cascade. Burges' cascade [7], [8] use a different reduced set of vectors for each classifier, while Romdhani [9] proposes a "greedy" cascade where every classifier uses the kernel evaluations of the previous ones. Other works [31] propose an hybrid approach by combining these two aforementioned ideas. These cascade schemes present an important improvement over the classification execution time, up to two orders of magnitude compared to the full SVM [31], while the RSV size can be significantly lower without any loss of the generalization performance [7].

This paper aims at adding a new dimension to the cascade classifier approach, which is the exploitation of the bit-precision utilized by each classifier in the cascade. The hypertile of the aforedescribed heterogeneous classifier presents an area cost, which is directly dependent on the problem's characteristics, the bit-precision of each attribute, and the dataset's dimensionality. This important quality enables the design of a cascade classifier, which implements each of the cascade units with a different numerical precision. Without loss of generality, let us assume a system with two cascade classifiers. The first classifier in the cascade is implemented in low-precision (LP), hence with smaller area cost and with larger throughput potential than the full-precision classifier, where the second classifier is implemented with higher precision, hence higher area cost, but with less demanding throughput requirements, as it processes only the data points that the first classifier cannot classify with certainty.

Certainly, the lower precision is expected to have some impact on the classification accuracy of the module. Thus, the LP classifier uses a more relaxed decision function f_{LP}, where two classification thresholds C_{cn} and C_{cp} are introduced, in

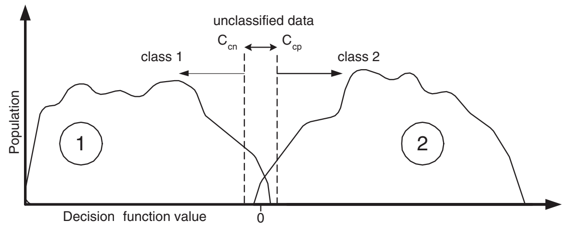


Fig. 6. Using the desired error rate to choose the thresholds C_{cn} and C_{cp}.

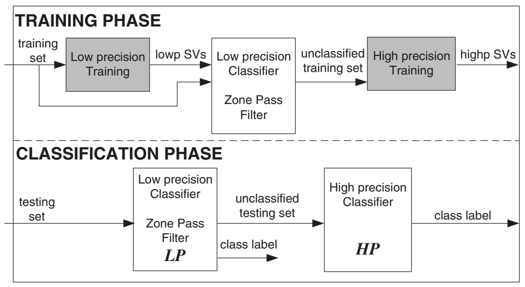


Fig. 7. Training and classification flow of the cascade classifier.

order to replace the sign function of (3). This relaxed decision function f_{LP} allows for labeling data whose classification product lies between the two thresholds as "unclassified" and passes them to the next classifier in the cascade chain. In that way, the decision function of the cascade classifier can be formulated as

$$F(\mathbf{x}) = \begin{cases} -1, & \text{if } f_{\text{LP}}(\mathbf{x}) \le C_{\text{cn}} \\ f_{\text{HP}}(\mathbf{x}), & \text{if } C_{\text{cn}} \le f_{\text{LP}}(\mathbf{x}) \le C_{\text{cp}} \\ 1, & \text{else} \end{cases} \quad (4)$$

where

$$f_{\text{LP}}(\mathbf{x}) = \sum_{i=1}^{|N_{\text{LP}}|} y_i \alpha_i K_{\text{LP}}(\mathbf{x}_i, \mathbf{x}) + b_{\text{LP}} \quad (5)$$

and

$$f_{\text{HP}}(\mathbf{x}) = \sum_{i=1}^{|N_{\text{HP}}|} y_i \alpha_i K_{\text{HP}}(\mathbf{x}_i, \mathbf{x}) + b_{\text{HP}}. \quad (6)$$

By choosing a targeted precision for the LP classifier, the desired false rate for the LP classifier indicates the appropriate thresholds C_{cn} and C_{cp}, as in Fig. 6. This figure shows the histogram of the kernel evaluations of the LP classifier. The thresholds C_{cn} and C_{cp} control the ratio a of the unclassified data to the overall problem size.

The decision function f_{LP} of the LP classifier LP uses the set of the SVs N_{LP}, obtained by training the SVM in LP. The high precision classifier (HP) uses a set of N_{HP} vectors and it is not trained over all the training set, instead, the LP classifier operates as a zone-pass filter in order to pass a reduced training set to the HP classifier. This decreases the training time of the HP classifier. Also, the HP training is focused on the data points which lie closest to the separating hyperplane, and the size N_{HP} is, hence, expected to be a subset of the LP one. The procedure flow for the cascade classifier is shown in Fig. 7. The LP classifier is first trained in LP over the entire training dataset. Then, it classifies the training set according to (4), but with training thresholds C_{tn} and C_{tp}. These thresholds

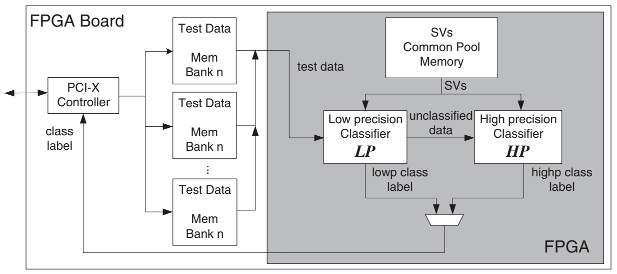


Fig. 8. Proposed cascade SVM classifier CC_{FIT}, in cases where the N_{SV} set fits in the FPGA internal memories.

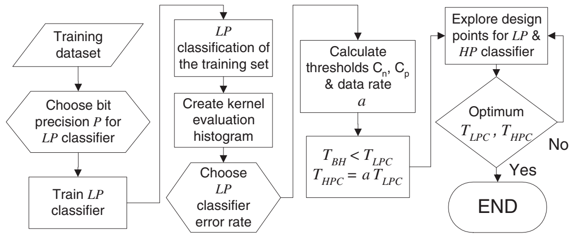

Training dataset
Choose bit precision P for LP classifier
Train LP classifier
LP classification of the training set
Create kernel evaluation histogram
Choose LP classifier error rate
Calculate thresholds C_n, C_p & data rate a
$T_{BH} < T_{LPC}$
$T_{HPC} = a\,T_{LPC}$
Explore design points for LP & HP classifier
Optimum T_{LPC}, T_{HPC}
No
Yes
END

Fig. 9. Design flow of the CC_{FIT} cascade classifier.

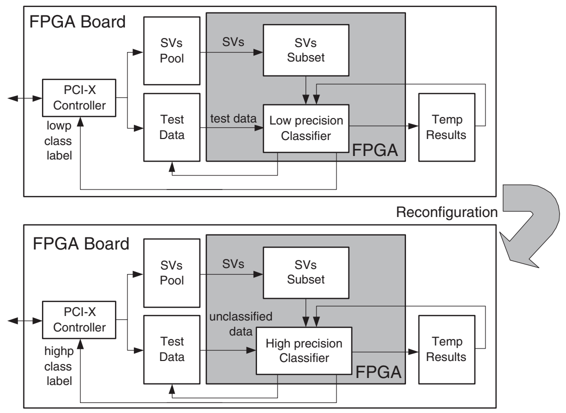


Fig. 10. Realizing the cascade SVM classifier CC_{RECONF} through FPGA reconfiguration.

control the size of the training set for the HP classifier. In the classification phase, the new data points are fed to the LP classifier, which use (4) and make a more relaxed classification decision. The selected false rate of the LP classifier determines the thresholds C_{cn} and C_{cp}, as in Fig. 6, but also the required throughput of the second classifier in the cascade chain.

The relationship between the capacity of the FPGA internal memories and the total size of the SV set $N_{\text{SV}} = N_{\text{HP}} \bigcup N_{\text{LP}}$ is an important constraint for the mapping of the cascade classifier and it infers the selection between two design choices, which are presented below.

B. *CC_{FIT}: Fitting the Cascade Chain in the FPGA Device*

In cases where the FPGA device offers adequate memory capacity for the entire N_{SV} set to be loaded at once, then both the LP and the HP classifier of the cascade scheme can potentially be mapped in the FPGA device. The proposed cascade scheme in this case will be referred as CC_{FIT} and its FPGA architecture is shown in Fig. 8.

Let us denote by A_{BC} the resource usage of the heterogeneous classifier, which is presented in Section III-C and will be referred as baseline heterogeneous classifier, for the purpose of this paper. A_{LP} and A_{HP} are the resource usages of the LP and the HP classifier in the cascade, respectively. By using the same resource budget for the baseline classifier and the cascade one

$$A_{\text{BC}} = A_{\text{LP}} + A_{\text{HP}} \tag{7}$$

the goal of the cascade classifier is to improve the classification time $t_{CC_{\text{FIT}}}$, compared to the execution time of the baseline heterogeneous classifier t_{BC}

$$t_{\text{BC}} > t_{CC_{\text{FIT}}} \Rightarrow \begin{cases} T_{\text{BC}} < T_{\text{LP}} \\ T_{\text{HP}} \geq aT_{\text{LP}}. \end{cases} \tag{8}$$

In other words, the objective is to attain a larger throughput T_{LP} for the LP classifier compared to the T_{BC} throughput of the baseline heterogeneous classifier. That means that the LP classifier requires more parallel hypertile instances than the initial heterogeneous approach, in order to increase its parallelization factor and throughput. At the same time, the HP classifier, which is not fully-unrolled, must have a throughput T_{HP} at least equal to aT_{LP}, where a is the ratio of the unclassified data to the overall problem size. The thresholds C_{cn} and C_{cp} are related to a, which control the throughput reduction for the HP classifier and the unrolling factor of its circuit. After the LP training phase, the kernel evaluation histogram is computed. The targeted throughput reduction a can then be chosen and the C_{cn} and C_{cp} threshold values are determined prior to the HP training phase. The (7), (8), and the ratio a specify some discrete design points for the LP and HP parallelization factors, and the design with the maximum overall throughput for the cascade classifier is found through an iterative process. The addition of an extra free parameter, such as a in the algorithm introduces a trade-off between higher efficiency and faster classification time against larger offline fine-tuning processing. The aforedescribed design flow for the cascade classifier is illustrated in Fig. 9.

C. *CC_{RECONF}: Employing the FPGA Reconfiguration*

There are cases where the SVM training of the upper half part of Fig. 7 produces a total N_{SV} set, which is larger than the maximum size of the internal FPGA memories. The N_{SV} set must then be split into more subsets. The LP classifier will produce and store intermediate results, until the projections on the last N_{SV} subset are performed and the final decision on the sign of (5) is evaluated. While the LP classifier computes intermediate results, it is preferable to dedicate all the available resources to the LP classifier, since the HP classifier input is not produced yet. At the end of the LP classification, the device is reconfigured and the HP classifier is mapped on the FPGA, in order to perform the SVM classification for the unclassified data points of the LP classifier. The temporary results are stored externally and fed back into the LP classifier for the projection processing of the next SV subset. The FPGA architecture of this proposed classifier scheme called CC_{RECONF} is illustrated in Fig. 10.

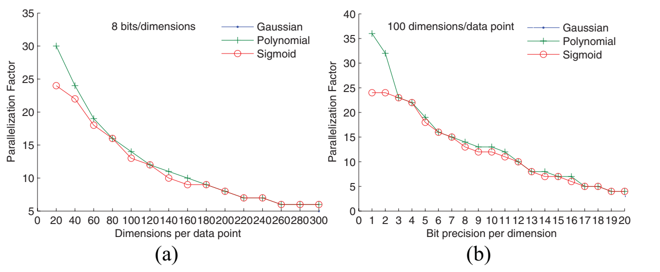


Fig. 11. (a) Parallelization factor scaling as a function to dimensionality and (b) bit-precision per feature for homogeneous problems.

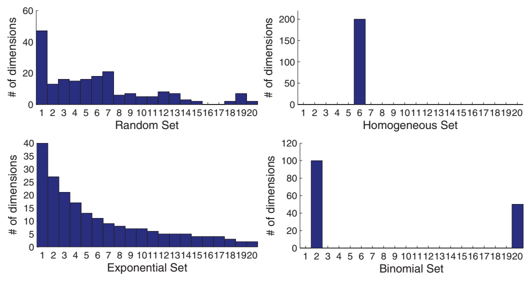


Fig. 12. Histograms of synthetic datasets.

Considering that both LP and HP classifiers use all the available FPGA resources in different configuration times, the constraint equations for the CC_{RECONF} classifier are

$$A_{\text{BC}} = A_{\text{LP}}, A_{\text{BC}} = A_{\text{HP}} \tag{9}$$

and the goal of the cascade classifier is to present a total classification time t_{CC} lower than the total classification time t_{BC} of the baseline heterogeneous classifier

$$t_{\text{BC}} > t_{CC_{\text{RECONF}}} \Rightarrow t_{\text{BC}} > t_{\text{LP}} + t_R + t_{\text{HP}} \tag{10}$$

where t_{LP}, t_{HP}, and t_R are the classification times of the LP classifier, the classification time of the HP classifier, and the reconfiguration time of the FPGA device, respectively. In this analysis, the classification times t_{BC}, t_{LP}, and t_{HP} include the time needed to load the internal memories with the SVs. For large-scale classification problems, the reconfiguration overhead is negligible compared to the actual processing times. The LP classifier parallelization factor is now maximized, since all the available resources can be used in LP hypertiles. This also applies for the HP classifier which, after reconfiguration, will have equal throughput to the baseline one. If the targeted precision for the LP classifier is b times lower than the baseline classifier, it can be derived from (10) that the following constraint equality on the SV load times lt_{BC}, lt_{CC} and the projection processing times pt_{BC}, pt_{CC} of the baseline, and the cascade classifier, respectively, exists

$$lt_{\text{BC}} + pt_{\text{BC}} > lt_{\text{CC}} + t_R + pt_{\text{CC}}. \tag{11}$$

In the worst-case scenario, where both the SV subsets of the LP and HP classifier are equal to the baseline classifier one, the total load time of the cascade classifier is $(1+\beta)\cdot lt_{BC}, \beta > 0$, which leads to

$$t_R < (1 - (\text{par}(\beta) - a) \cdot pt_{\text{BC}} - \beta \cdot lt_{\text{BC}} \tag{12}$$

where $\text{par}(\beta)$ is the ratio between the parallelization factor of the BC classifier and the LP classifier, and depends on the resource constraints and the dataset's characteristics. It is easily derived that the chosen precision for the LP classifier and the throughput reduction a specify some discrete design points and the reconfiguration time t_R can be negligible, when the classification problem is large-scaled and the processing time pt_{BC} for the projections is high.

The design flow of the CC_{RECONF} classifier is similar to the one of the CC_{FIT}, which is presented in Fig. 9. The only difference is that the constraint equations (7), (8), and replaced by (9) and (12). The employment of reconfiguration allowed for the extension of the design space, offering more design options for the cascade scheme. The choice between the CC_{FIT} and CC_{RECONF} cascade options is determined by comparing their performances according to (8) and (11).

V. Implementation Results

A. Performance of the Baseline Heterogeneous SVM Classifier

The targeted device for the proposed architecture was the Altera's Stratix III EP3SE260. The results can be easily expanded to other targeted devices by changing the resource constraints of the design flow. The architecture is captured in VHDL and the floating-point modules are generated by the Altera tools and the Altera floating-point compiler [32]. The targeted operating frequency of all produced designs ranges between 200–250 MHz.

It should be noted that all the results demonstrated in the figures henceforth are actual points of the design space, constrained by the targeted device. Nevertheless, the proposed architecture is fully-scalable and the implementation results can easily be projected to any other resource constraint problem, simply by targeting another device. The left graph of Fig. 11 illustrates the scaling of the achieved parallelization factor of the heterogeneous architecture as a function to the bit-precision of a 100-feature homogeneous dataset, for all targeted kernels. The parallelization factor expresses the number of parallel hypertiles that can be instantiated for the heterogeneous classifier on the targeted FPGA. On the right graph, Fig. 11 presents the scaling of the parallelization factor as a function to a homogeneous problem's dimensionality with eight-bit-precision per attribute, which is common to image processing applications. The parallelization factor scales in an exponential-like fashion function to the dimensionality, while dropping almost linearly with the increase of the bit-precision.

The heterogeneous architecture was evaluated for a set of synthetic datasets that have the same number of total bits per datum $\mathbf{x}$ but different precision distribution among their features. The histograms of these synthetic datasets are shown in Fig. 12. The horizontal axis in these figures is the required precision in bits, ranging between 1–20, while the vertical axis measures the number of attributes in the dataset that requires the specific bit-precision. It is obvious that the distribution of bit-precision among the attributes range in different ways, however, the total sum of bits required for the datum representation is equal among all datasets. The proposed

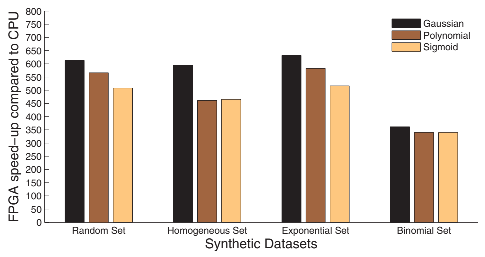


Fig. 13. Achieved speed-up for synthetic datasets.

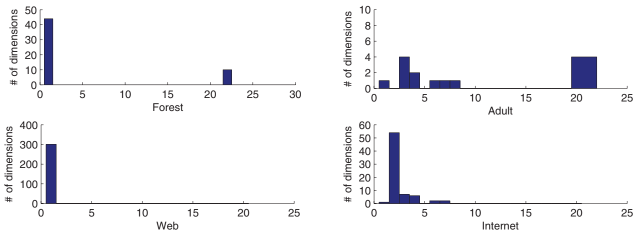


Fig. 14. Histograms of real-world datasets.

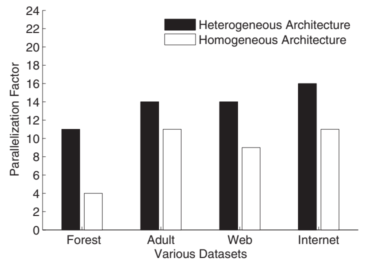


Fig. 15. Achieved parallelization factor for real-world datasets.

FPGA classifier was compared to a *C* implementation on a PC with a 3-GHz Intel CORE 2 DUO and 2 GB of RAM. Fig. 13 shows that the proposed FPGA classifier presents a speed-up of two orders of magnitude. It can be derived that, even when the total bits of all the attributes per data point are equal, the dynamic range distribution among the features can offer significant reduction in the resource utilization of the BC classifier and, hence, important throughput gain.

The heterogeneous datasets of Fig. 12 uncover the true potential of the proposed heterogeneous classifier. The FPGA accelerator in [18], as well as the GPU works of [24] and [25], treat heterogenous attributes like the ones of the random dataset or the exponential in a homogeneous manner. The work in [18] represents all the attributes of the random dataset with 20 bits, which is the largest precision observed among the various attributes, while the GPU works in [24] and [25] use single floating-point arithmetic for all these dimensions. It is therefore obvious that the available hardware resources are not utilized in the most efficient way. Fig. 13 shows that the proposed heterogeneous classifier succeeds in dedicating hardware resources with respect to the dynamic range requirements of the dataset attributes, hence instantiating efficient processing elements with larger parallelization potentials.

Fig. 14 presents the bit-precision histograms of some popular classification datasets [5]. Forest Covertype has 581-K data points of 54 attributes, Adult's size is 32 K with dimensionality $D = 14$, Web is a 300-binary attribute dataset of 49-K points and Internet Usage Data is a dataset with 72 categorical and binary attributes, of total 10-K instances. Fig. 15 captures the significant improvement in the parallelization factor of the proposed heterogeneous architecture, when compared against a homogeneous approach, like the one presented in [17]. The parallelization factor expresses the number of parallel instances of classifier hypertiles and it is a direct factor of the achieved speed-up. In most cases, the heterogeneous architecture outperforms the homogeneous one by a 2–3 factor. The heterogeneous architecture outperforms the homogeneous one in all problems with wide distributions of their attributes' precision requirements. This is derived by the fact that, due to the MAC unit implementation for the inner products, the homogeneous architecture treats all the dataset attributes in the same way. The different precision requirements of the dataset attributes are ignored, and the MAC units of the homogeneous architecture need to be large enough to satisfy the computational precision for the multiplication of the worst-case bit precision among all attributes. On the other hand, the proposed heterogeneous classifier treats the attributes individually and uses the minimum bit-precision required for each attribute multiplication, resulting in an area-efficient mapping.

The proposed heterogeneous classifier is compared to previous FPGA and GPU works [18], [24], [25] on the MNIST [4] dataset. The size of testing set is 10-K samples. MNIST includes ten different classes, so the problem here is formulated as a classification task between class 2 versus all the others. Comparing the hardware accelerators under this homogeneous benchmark does not demonstrate the dynamic range exploitation of the proposed heterogeneous classifier, which is studied in the previous paragraph. However, the benefits of using all the available heterogeneous resources of a modern FPGA device will come clear. A previous FPGA work [18] reports a raw computational speed of the SVM classification core of 40 GMACs on a Xilinx Virtex-5 LX330T. The targeted Altera device has slightly less number of registers but 4 × more DSPs than the FPGA in [18]. The proposed heterogeneous classifier architecture presents a 8.1 × speed-up over the homogeneous work in [18]. In [18], the dot-product computations are implemented only with DSP units and the available logic resources are not exploited, along with the bit-precision requirements of the targeted problem. The works on GPUs [24], [25] formulate the multiclass MNIST problem as an even-versus-odd digit classification. On a 2.5-K classification set, [24] reports an execution time of 1.98 s, while [25] needs 8.4 s to classify 10-K vectors. The proposed heterogeneous architecture achieves 7.8 × and 8.55 × speed-ups compared to [24] and [25], respectively.

B. Cascade Classifier versus the Baseline Classifier

In this section, the proposed cascade classifier architecture is compared to the proposed baseline heterogeneous classifier on the MNIST [4] dataset. The comparative results between the baseline heterogeneous classifier and the cascade classifier are shown in Table II. Using a Gaussian kernel, the full-precision

TABLE II
PERFORMANCE COMPARISON OF BASELINE AND CASCADE CLASSIFIER ON MNIST DATASET

Classifier architecture	Size of SV set	Classification accuracy	Speed-up over CPU
Heterogenous classifier (Section III)	9438	98.96%	480.14
Cascade classifier (Section IV)	9798/6492	98.95%	1098.59

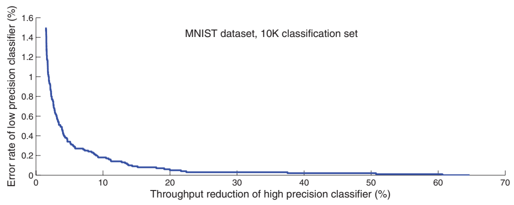


Fig. 16. Function of the LP error rate to the HP throughput reduction a.

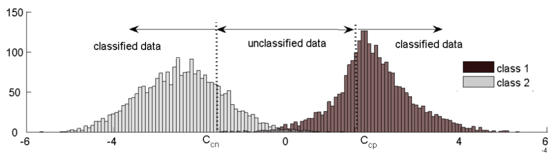


Fig. 17. Choosing the thresholds of the LP classifier through the kernel evaluation histogram.

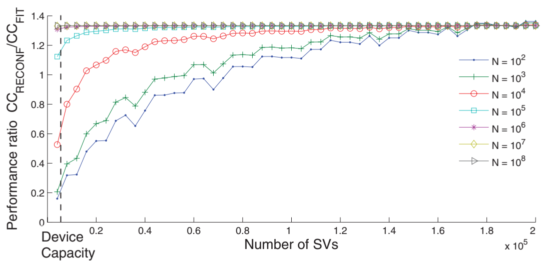


Fig. 18. Performance ratio $CC_{\text{RECONF}}/CC_{\text{FIT}}$ for a homogeneous classification dataset on the Altera's Stratix III EP3SE260 device.

training identifies 9438 SVs and the classification accuracy is 98.96%. The baseline heterogeneous architecture presents a parallelization factor of 2 for the targeted device, and the overall speed-up is 480 $\times$ compared to the CPU classification.

For the specific dataset, the precision for the LP classifier was set to four bits and the SVM training returns 9798 SVs. In the classification phase, the desired error rate for the LP classifier is 0.04%. This selection gives the training thresholds $\text{C}_{\text{tn}} = -1.9 \times 10^{-4}$ and $\text{C}_{\text{tp}} = 1.9 \times 10^{-4}$. Using these values, the training set was fed to the LP classifier and 20 779 data points were unclassified. These data points compose the set for the HP training, which identified 6492 SVs.

Fig. 16 shows the relationship between the LP classifier error rate and the throughput reduction a. The error rate is the ratio of the misclassified data to the overall classification size. The ratio a is the percentage of the unclassified vectors, which will have to be propagated to the next cascade level and it expresses the unrolling factor of the HP classifier circuit. For the desired error rate, $a = 23.57\%$ and hence, according to (5), the throughput of the HP classifier must be at least 0.2357 $\times$ of the LP one. Fig. 17 shows the histogram of the LP kernel evaluations. The desired error rate gives $C_{\text{cn}} = -1.56 \times 10^{-4}$ and $C_{\text{cp}} = 1.56 \times 10^{-4}$. The classification accuracy of the first cascade level is 99.96%, and 2357 vectors were unclassified and propagated to the higher-precision level. The accuracy of the HP classifier over this smaller set was 95.67%. The overall accuracy for the cascade classifier is 98.85%, resulting in a negligible loss of the generalization classification performance.

The resource budget of the targeted device allows for one HP and four parallel LP heterogeneous classifiers. The HP classifier of the cascade is half-rolled, utilizing 50% of the resources of the fully-unrolled classifier. The performance of the cascade classifier was doubled, while targeting the same device as the baseline heterogeneous classifier. It can be derived by Fig. 16 that the relationship between the selected false rate of the LP classifier drops exponentially with the increase of the data rate a. If the user was eager to accept a front classification false rate of 0.5%, then $a = 0.0450$, which implies considerable savings in the HP resource usage.

C. Comparison of the Proposed Cascade Classifier Schemes

In Section IV, two design choices were presented for the cascade SVM classifier. When the FPGA memory capacity allows for one-load of the SV set in the internal FPGA memories, the CC_{FIT} scheme can be used. Otherwise, employing the FPGA's reconfigurability seems to be the best feasible choice. The throughput performance of these two design choices is compared on the Gaussian classification task of a synthetic homogeneous dataset with 300-D and eight-bit/dimension, resulting in 2400 bits per SV. Choosing the desired throughput reduction of the HP classifier at $a = 14\%$, the performance ratio between the CC_{RECONF} architecture and the CC_{FIT} architecture is shown in Fig. 18, for a wide range of the SV size and for seven different sizes of classification problems. The SV set size N_{HP} for the HP classifier is half the N_{LP} size of the LP one. The CC_{FIT} is designed with seven parallel hypertiles for the LP classifier and one full-precision hypertile, which is enough to satisfy the $0.14 \times T_{\text{LP}}$ throughput constraint, according to (8). On the other hand, the LP and HP classifiers of the CC_{RECONF} cascade scheme can use all the available resources of the FPGA device, since they are mapped on it during different configuration times. The parallelization factors of the LP and HP classifiers are ten and five, respectively, as they can be derived from Fig. 11.

In Fig. 18, it can be observed that the reconfiguration time penalty becomes insignificant, as the sizes of the classification problem and the SV set increase. Moreover, for a specific SV size, the performance - in terms of classification times - of the CC_{RECONF} classifier improves over the CC_{FIT} performance as the classification problem size N increases, CC_{RECONF} even outperforms CC_{FIT} for large N sizes. The vertical dashed line represents the limit of the SV population that the internal memories of the targeted device can store. The CC_{FIT} performance on the right side of the dashed line is only a theoretical

TABLE III
PERFORMANCE COMPARISON OF VARIOUS HARDWARE ACCELERATORS ON MNIST DATASET

Classifier architecture	Speed-up
FPGA classifier [18]	1×
GPU classifier [24]	1.1×
GPU classifier [25]	1.18×
FPGA heterogenous classifier (Section III)	8.1×
FPGA cascade classifier CC_{FIT} (Section IV-B)	18.57×
FPGA cascade classifier CC_{RECONF} (Section IV-C)	25.4×

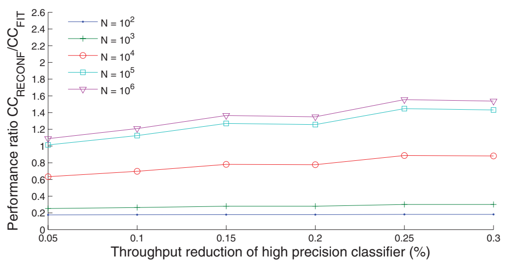


Fig. 19. Performance ratio $CC_{\text{RECONF}}/CC_{\text{FIT}}$ function to throughput reduction a on the Altera's Stratix III EP3SE260 device.

projection, as if the FPGA memories could fit such SV sizes. There are several modern FPGA devices with larger memory capacity than the Stratix III EP3SE260, and the dashed line could move further to the right, allowing for the CC_{FIT} to outperform the CC_{RECONF} scheme for more case studies. The CC_{FIT} presents much lower classification times then the reconfiguration scheme, from 1.25 to 5 times, for SV set sizes up to 50 K. It is obvious, however, that, when targeting very large-scale problems, the CC_{RECONF} scheme is the best option. Moreover, when the size of the SVs set exceeds the memory capacity of the targeted FPGA device (the right side of the dashed device capacity line), the CC_{RECONF} classifier is the only available option, since the fully unrolled. The CC_{FIT} classifier cannot fit and be implemented in the FPGA device. The observed maximum speed-up of 1.37× is dependent on the SV size, the relationship of the parallelization factors for the LP and HP classifiers of the two cascade schemes. Hence, it varies subject to the dataset characteristics and the values of a and β. The significance of this speed-up is fully revealed when multiplied by the speed-up already achieved by the CC_{FIT} classifier over the CPU. Table III shows the comparative speed-up factors between the proposed heterogeneous classifier, the proposed cascade classifier schemes, and other hardware-based accelerators, using the MNIST [4] dataset as a benchmark.

Fig. 19 shows how the classification time ratio between the two cascade options ranges function to the threshold reduction a of the HP classifier. The performance of CC_{FIT} and CC_{RECONF} is measured over a range of different classification set windows of size N, for the cases where the SV set size is within the capacity range of the FPGA device. What is interesting here is that, by increasing the throughput of the HP classifier, the performance of the CC_{FIT} scheme worsens. For small values of a, the thresholds C_{cn} and C_{cp} in Fig. 6 tighten, thus decreasing the throughput of the HP classifier in the cascade. This allows for more resource usage on the LP classifier and higher front-end parallelization. Hence, there is a trade-off between the throughput reduction a, the LP classification accuracy, and the performance of the CC_{FIT} cascade scheme. On the other hand, the throughput rise of the HP classifier improves the classification performance, since the thresholds C_{cn} and C_{cp} are wider. However, the LP parallelization factor of the CC_{FIT} scheme worsens, because there are less discrete design points available to satisfy the constraint equation (8). The CC_{RECONF} scheme is not influenced by a, since each classifier in the cascade causes all the FPGA resources.

The main advantage of the CC_{FIT} classifier scheme is the superior performance in relatively small classification datasets, where the reconfiguration penalty biases the CC_{RECONF} performance. On the other hand, the CC_{RECONF} scheme is the only solution when the resource constraints do not allow for CC_{FIT} fitting, while its performance improves with the size of classification window set. For streaming applications, the LP classification task of the LP classifier introduces a delay proportional to N and the class labels are fed back in bursts, the overall performance of the application though is slightly increased compared to the CC_{FIT} scheme.

The implementation results highlight the performance gain of the proposed cascade classifier. The exploitation of the relationship between the bit-precision and the resource utilization of the heterogeneous architecture made it possible to implement a cascade scheme of custom-precision classifiers. By choosing an arbitrary lower precision than the required one, the classification accuracy of the classifier drops. However, if this LP classifier is used as the first level of a cascade scheme, it increases the throughput of the system by exploiting its larger parallelization potential. The HP classifier can be implemented using significantly less resources than a baseline heterogeneous one, because its throughput only needs to satisfy the traffic of unclassified data coming from the LP classifier. The cascade classifier achieved the same classification accuracy as the normal one and boosted the already high performance of the heterogeneous classifier, under the same resource constraints. The main contribution of the CC_{RECONF} classifier is the expansion of the potential design space and the ability to target classification problems, whose resource utilization requirements exceed the available resources of the FPGA device. Moreover, in cases where the size of the SV set exceeds the memory capacity of the targeted FPGA device, the reconfiguration cascade scheme can perform equally high and in some cases even better than the CC_{FIT} classifier. For cases where the original SV set fits in the FPGA device but the CC_{FIT} SV size does not, the heterogeneous BC classifier still is the preferred choice. However, in cases like these, if the classification load is heavy, the CC_{RECONF} classifier becomes more preferable than the heterogeneous BC one.

VI. CONCLUSION

This paper presented a fully scalable heterogeneous FPGA cascade classifier for the acceleration of the SVM classification. By exploiting the dynamic range diversities among the training problem features, the word-length optimizations in the fixed-point domain allow for efficient usage of the available resources by the classifier processing unit. The logic versus DSP usage ratio was maintained to the targeted device's one and the resource utilization was well-balanced, resulting in high parallelization factors. The proposed heterogeneous architecture achieved to speed-up the CPU classification execution time by 2–3 orders of magnitude, while outperforming over 7 $\times$ other works on FPGAs and GPUs.

The novel cascade classifier scheme exploits the characteristics of the heterogeneous architecture, and results in a hardware-friendly approach with even higher-performance. The cascade classifier takes advantage of the relationships between custom precision, resource utilization, and classification accuracy, in order to increase the system throughput. It was shown that, by investigating the possible design points, the proposed cascade classifier doubled the throughput of the application under the same resource constraints, with an insignificant penalty on the overall classification performance. Also, it was proven that a reconfigurable cascade scheme can switch from low- to high-precision classification and maintain the performance in high-levels, when the FPGA resource constraints do not allow for a fully-unrolled cascade classifier.

The proposed heterogeneous FPGA classifier expresses the diversity of the dataset dynamic ranges into customized circuitry through word-length optimizations, thus boosting the parallelization potential. Taking this a step further, the cascade architecture adds an extra degree-of-freedom to the problem by exploiting the relationship between the bit-precision and the classification accuracy.

References

[1] V. N. Vapnik, *The Nature of Statistical Learning Theory*. Berlin, Germany: Springer-Verlag, 1995.

[2] H. Byun and S.-W. Lee, "Applications of support vector machines for pattern recognition: A survey," in *Proc. 1st Int. Workshop Pattern Recognit. Support Vector Mach.*, 2002, pp. 213–236.

[3] J.-C. Terrillon, M. N. Shirazi, M. Sadek, H. Fukamachi, and S. Akamatsu, "Invariant face detection with support vector machines," in *Proc. 15th Int. Conf. Pattern Recognit.*, vol. 4. 2000, pp. 210–217.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[5] A. Asuncion and D. Newman. (2007). *UCI Machine Learning Repository* [Online]. Available: http://www.ics.uci.edu/∼mlearn/MLRepository.html

[6] L. Mandrake, K. Wagstaff, D. Gleeson, U. Rebbapragada, D. Tran, R. Castano, S. Chien, and R. Pappalardo, "Onboard detection of natural sulfur on a glacier via an SVM and hyperion data," in *Proc. 30th IEEE Aerosp. Conf.*, Mar. 2009, pp. 1–4.

[7] C. J. Burges, "Simplified support vector decision rules," in *Proc. Int. Conf. Mach. Learn.*, 1996, pp. 71–77.

[8] C. J. Burges and B. Schölkopf, "Improving the accuracy and speed of support vector machines," in *Advances in Neural Information Processing Systems 9*. Cambridge, MA: MIT Press, 1997, pp. 375–381.

[9] S. R. M. Ratsch and T. Vetter, "Efficient face detection by a cascaded support vector machine expansion," *Proc. Royal Soc. London Ser.*, vol. 460, no. 2051, pp. 3283–3297, 2004.

[10] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods: Support Vector Learning*. Cambridge, MA: MIT Press, 1999, pp. 185–208.

[11] T. Joachims, "Transductive inference for text classification using support vector machines," in *Proc. 6th Int. Conf. Mach. Learn.*, 1999, pp. 200–209.

[12] T. Joachims, "Training linear SVMs in linear time," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2006, pp. 217–226.

[13] S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. Murthy, "A fast iterative nearest point algorithm for support vector machine classifier design," *IEEE Trans. Neural Netw.*, vol. 11, no. 1, pp. 124–136, Jan. 2000.

[14] S. Martin, "Training support vector machines using Gilbert's algorithm," in *Proc. 5th IEEE Int. Conf. Data Mining*, Washington, DC, Nov. 2005, pp. 306–313.

[15] J. Mercer, "Functions of positive and negative type and their connection with the theory of integral equations," *Phil. Trans. Royal Soc. London*, vol. 209, nos. 441–458, pp. 415–446, 1909.

[16] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA: MIT Press, 2001.

[17] M. Papadonikolakis and C.-S. Bouganis, "A scalable FPGA architecture for non-linear SVM training," in *Proc. Int. Conf. FPT Technol.*, Dec. 2008, pp. 337–340.

[18] S. Cadambi, I. Durdanovic, V. Jakkula, M. Sankaradass, E. Cosatto, S. Chakradhar, and H. Graf, "A massively parallel FPGA-based coprocessor for support vector machines," in *Proc. 17th IEEE Symp. Field Programm. Custom Comput. Mach.*, Apr. 2009, pp. 115–122.

[19] D. Anguita, A. Boni, and S. Ridella, "A digital architecture for support vector machines: Theory, algorithm, and FPGA implementation," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 993–1009, Sep. 2003.

[20] F. Khan, M. Arnold, and W. Pottenger, "Hardware-based support vector machine classification in logarithmic number systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 5. May 2005, pp. 5154–5157.

[21] K. Irick, M. DeBole, V. Narayanan, and A. Gayasen, "A hardware efficient support vector machine architecture for FPGA," in *Proc. Annu. IEEE Symp. Field-Programm. Custom Comput. Mach.*, Apr. 2008, pp. 304–305.

[22] M. Ruiz-Llata and M. Yèbenes-Calvino, "FPGA implementation of support vector machines for 3D object identification," in *Proc. 19th Int. Conf. Artif. Neural Netw. I*, 2009, pp. 467–474.

[23] C. Hsu, M.-K. Ku, and L.-Y. Liu, "Support vector machine FPGA implementation for video shot boundary detection application," in *Proc. IEEE Int. SOC Conf.*, Sep. 2009, pp. 239–242.

[24] B. Catanzaro, N. Sundaram, and K. Keutzer, "Fast support vector machine training and classification on graphics processors," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 104–111.

[25] A. Carpenter. (2009). *CUSVM: A Cuda Implementation of Support Vector Classification and Regression* [Online]. Available: http://patternsonascreen.net/cuSVM.html

[26] NVidia. (2008). *NVIDIA CUDA Compute Unified Device Architecture, Programming Guide*, Santa Clara, CA [Online]. Available: http://www.nvidia.co.uk/cuda

[27] M. Papadonikolakis, C.-S. Bouganis, and G. Constantinides, "Performance comparison of GPU and FPGA architectures for the SVM training problem," in *Proc. Int. Conf. Field-Programm. Technol.*, 2009, pp. 388–391.

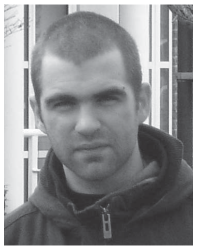
[28] E. G. Gilbert, "An iterative procedure for computing the minimum of a quadratic form on a convex set," *SIAM J. Control*, vol. 4, no. 1, pp. 61–80, 1966.

[29] M. Papadonikolakis and C. Bouganis, "A novel FPGA-based SVM classifier," in *Proc. Int. Conf. Field-Programm. Technol.*, Dec. 2010, pp. 283–286.

[30] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to support vector classification," Dept. Comput. Sci., Nat. Taiwan Univ., Taipei, Taiwan, Tech. Rep. 1, 2003.

[31] I. Kukenys and B. McCane, "Classifier cascades for support vector machines," in *Proc. 23rd Int. Conf. Imag. Vis. Comput.*, Nov. 2008, pp. 1–6.

[32] M. Langhammer, "Floating point datapath synthesis for FPGAs," in *Proc. Int. Conf. Field Programm. Logic Appl.*, Sep. 2008, pp. 355–360.



Markos Papadonikolakis received the Diploma degree in electrical engineering and computer technology from the University of Patras, Patras, Greece, in 2006. He is currently pursuing the Ph.D. degree with the Circuits and Systems Group, Department of Electrical and Electronic Engineering, Imperial College London, London, U.K.

He joined Imperial College London in 2007. His current research interests include field programmable gate arrays and reconfigurable computing, hardware designs, parallel processing, machine learning, and image processing.

Christos-Savvas Bouganis (S'01–M'03) is a Lecturer with the Electrical and Electronic Engineering Department, Imperial College London, London, U.K. He has published over 30 research papers in peer-referred journals and international conferences, and he has contributed three book chapters. His current research interests include the theory and practice of reconfigurable computing and design automation, mainly targeting digital signal processing algorithms.

He currently serves on the program committees of many international conferences, including FPL, FPT, DATE, SPPRA, and VLSI-SoC. He is an Editorial Board Member of the IET Computers and Digital Techniques and the *Journal of Systems Architecture*. He has served as the General Chair of ARC in 2008 and the Program Chair of the IET FPGA designers' forum in 2007.