Embedded Hardware-Efficient Real-Time Classification With Cascade Support Vector Machines

Christos Kyrkou, *Member, IEEE*, Christos-Savvas Bouganis, *Member, IEEE*, Theocharis Theocharides, *Senior Member, IEEE*, and Marios M. Polycarpou, *Fellow, IEEE*

***Abstract*— Cascade support vector machines (SVMs) are optimized to efficiently handle problems, where the majority of the data belong to one of the two classes, such as image object classification, and hence can provide speedups over monolithic (single) SVM classifiers. However, SVM classification is a computationally demanding task and existing hardware architectures for SVMs only consider monolithic classifiers. This paper proposes the acceleration of cascade SVMs through a hybrid processing hardware architecture optimized for the cascade SVM classification flow, accompanied by a method to reduce the required hardware resources for its implementation, and a method to improve the classification speed utilizing cascade information to further discard data samples. The proposed SVM cascade architecture is implemented on a Spartan-6 field-programmable gate array (FPGA) platform and evaluated for object detection on 800 × 600 (Super Video Graphics Array) resolution images. The proposed architecture, boosted by a neural network that processes cascade information, achieves a real-time processing rate of 40 frames/s for the benchmark face detection application. Furthermore, the hardware-reduction method results in the utilization of 25% less FPGA custom-logic resources and 20% peak power reduction compared with a baseline implementation.**

***Index Terms*— Cascade classifier, field-programmable gate array (FPGA), local binary pattern (LBP), neural networks (NNs), parallel architectures, real-time and embedded systems, support vector machines (SVMs).**

I. Introduction

SUPPORT vector machines (SVMs) [1] constitute a powerful set of machine learning algorithms, which have been utilized in a wide range of classification applications, demonstrating high classification accuracies [2], [3]. The classification complexity of SVMs is proportional to the number of training samples needed to specify the separating hyperplane between classes, referred to as SVs. Hence, for large-scale problems, the high classification accuracy rates demonstrated by SVMs come at the cost of increased computational complexity. As such, when considering embedded applications (e.g., embedded vision, automotive, and security) with real-time online classification requirements and power-consumption constraints and limited resources and area, the design of SVM-based classification systems with hundreds of SVs and a large number of instances that need to be classified becomes difficult. Heisele *et al.* [4], Kukenys and McCane [5], and Ma and Ding [6] proposed a cascaded classification scheme in order to speed up the SVM classification process for a class of the aforementioned applications, such as embedded object detection, where the majority of the data that need to be classified belong to one of the two classes. Under this scheme, multiple SVMs are arranged in stages of increasing computational complexity as well as accuracy. The early stages, which are computationally less demanding, are tasked with the removal of a large amount of negative class data, so that the latter stages, which have higher accuracy and thus higher computational complexity, only classify the samples that successfully pass the previous stages. Hence, using the cascade approach results in significant speedups over monolithic (single) SVM classification [4], [6]. However, online real-time classification on resource-constraint embedded systems, which need low-power operation, is still challenging to achieve, especially for large-scale streaming data problems, such as video object detection [4].

This has motivated a lot of research toward accelerating SVMs using parallel computing platforms, such as graphics processing units (GPUs) [7] and field-programmable gate arrays (FPGAs) [8]–[10]. Implementations of SVMs on GPU platforms have been proposed recently, however, GPUs face challenges with regard to power consumption [11] and thus it is difficult to deploy them in embedded environments. Hence, at present, FPGAs and customized hardware accelerators that consume less power and can be built into small systems offer an attractive platform for embedded applications. Existing SVM hardware architectures consider monolithic SVM classifiers, which are not optimized to handle problems

Manuscript received June 30, 2014; revised April 20, 2015; accepted April 21, 2015. Date of publication May 18, 2015; date of current version December 17, 2015. This work was supported by the European Research Council Advanced Grant through the Fault-Adaptive Project under Grant 291508.

C. Kyrkou, T. Theocharides, and M. M. Polycarpou are with the KIOS Research Center for Intelligent Systems and Networks, Department of Electrical and Computer Engineering, University of Cyprus, Nicosia 1678, Cyprus (e-mail: kyrkou.christos@ucy.ac.cy; ttheocharides@ucy.ac.cy; mpolycar@ucy.ac.cy).

C.-S. Bouganis is with the Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2AZ, U.K. (e-mail: 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.2015.2428738

2162-237X © 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

where the majority of the data belong to one of the two classes. As such, designing hardware architectures for multistage cascade SVMs based on existing approaches are a challenging task due to the increase in the number of classifiers, and their different computational complexities.

In this paper, we propose a specialized hardware architecture and design approaches for embedded online cascade SVM classification applications, such as real-time video object detection, where classification needs to be performed in real time, with low power, and often with limited available resources. The presented design methodologies extend and improve our preliminary research in [12], which showed the advantages of a cascade hardware SVM over a monolithic hardware SVM. In this paper, we provide further details on the optimized cascade hardware architecture, which can facilitate high frame rates. We also show how to reduce the hardware complexity of cascade SVMs, which is based on rounding off the SVM training data to the nearest power of two values, in order to improve both area and power while maintaining the accuracy by replacing multiplication with shift operations. Moreover, a novel approach is introduced that improves the frame rate by reducing the number of samples that reach the more computationally demanding stages through a neural network (NN) that evaluates preceding cascade stage responses.

The proposed architecture and methods are implemented as part of a complete online video classification system on a Spartan-6 FPGA platform. The system was evaluated on a larger test set and higher resolution images (800×600) than our prior work using face detection as the embedded benchmark application. The system achieves 40 frames/s, which is capable for real-time processing while processing more windows than other works, and an 80% detection accuracy, which is on par with cascade SVM software implementations for the targeted application. Furthermore, the hardware-reduction method resulted in the utilization of 25% less FPGA logic resources and reduction of peak power by 20%, with only a 1% reduction in classification accuracy.

This paper is organized as follows. Section II provides the background on SVMs, cascade classifiers, and related work. Section III details the hardware architecture for cascade SVM processing, the hardware-reduction method, as well as the cascade response evaluation process. Section IV presents FPGA-based experimental results as well as comparison with related works. Finally, the conclusion is drawn in Section V.

II. Background

A. Support Vector Machines

An SVM is a supervised binary classification algorithm, which maps data into a high-dimensional space, where an optimal separating hyperplane is constructed [1]. SVMs are presented with a training set consisting of pairs of data samples x_i, and class labels y_i (-1 for negative and 1 for positive samples), and try to find a mapping function f, such that $f(x_i) = y_i$ for sample i in the training set. This function captures the relationship between the data samples and their respective class labels. An SVM separates the data

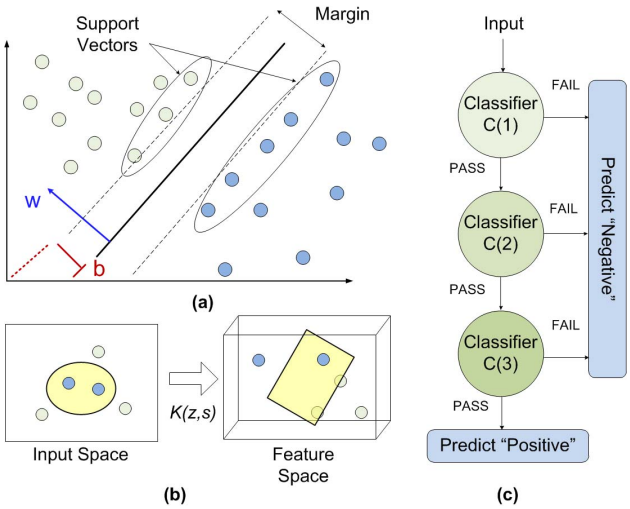


Fig. 1. (a) SVM concepts: separating hyperplane, SVs, normal vector w, bias, and margin. (b) Kernel trick visualization. (c) Cascade classification scheme overview.

samples of two different classes, by finding the hyperplane with the maximum margin from the data samples that lie at the boundary of each class [Fig. 1(a)]. The class samples that are on the boundary are called SVs and influence the formation of the hyperplane [1], [2]. The SVs are obtained during the SVM training phase, and correspond to nonzero alpha coefficients derived from the training optimization problem [2], and constitute the SVM classification model with which new input data are classified. In many real-world applications, the data samples may not be linearly separable. SVMs utilize a technique called the kernel trick [2], to project the data into higher dimensional space where linear separation is possible and then proceed to find the decision surface. This formulation allows projecting data into a higher dimensional space, where linear separation is possible [Fig. 1(b)], though a kernel function $K(x_i, x_j) = \varphi(x_i)\varphi(x_j)$, without the need to explicitly use a mapping function φ. Overall, the classification decision function for SVMs is given in (1), where N_{sv} is the number of SVs obtained from training, α_i are the alpha coefficients, y_i are the class labels of each sample, s_i are the SVs, z is the input vector, $k(z, s_i)$ is the chosen kernel function, and b is the bias

$$\text{CDF}(z)\text{: sign}\left(\sum_{i=1}^{N_{\text{SV}}} \alpha_i y_i K(z, s_i) + b\right). \tag{1}$$

The computational demands of SVM classifiers depend on the choice of kernel function, the most common of which are illustrated as follows:

$$\text{Linear (Dot Product): } K(z, s) = (z \cdot s) \tag{2}$$

$$\text{Polynomial: } K(z, s) = ((z \cdot s) + \text{const})^d, \quad d > 0 \tag{3}$$

$$\text{Radial Basis Function: } K(z, s) = \exp(-\|z - s\|^2/2\sigma^2). \tag{4}$$

The linear kernel (2) for SVMs corresponds to a dot-product operation between the input data and a feature vector w, which is the decision hyperplane normal vector [Fig. 1(a)],

and is computed directly from the SVs using $w = \sum_{i=1}^{N_{sv}} y_i \alpha_i s_i$. However, in the case of nonlinear SVMs (3) and (4), the kernel is a more complex function and the feature vector cannot be directly obtained from the SVs. Hence, the input vector needs to be processed with all SVs, and the kernel-specific operations need to be performed, before a classification outcome can be obtained. To reduce the computational demands of nonlinear kernels, a number of techniques have been proposed. One such method is the reduced-set-method [13], which tries to find a smaller set of vectors, called reduced-set-vectors (RSVs), in order to approximate the decision function of the full SVM retaining most of the classification capabilities [6], which yields an RSV machine.

B. Cascade Support Vector Machines

In many real-world problems, nonlinear kernels are necessary in order to obtain accurate classification results on complex data sets. However, classification rates can be slow with such kernels as they produce many SVs that need to be processed per input sample. In this paper, we focus on the acceleration of SVM-based classification for a certain class of applications, such as image object classification, that exhibit the following characteristics: 1) the majority of the samples presented to the classifier belong to the negative class and 2) the majority of negative samples do not exhibit similar features to positive samples. Software implementations in [14] and [15] have tried to take advantage of these two observations utilizing stages of SVMs of increasing complexity, which are sequentially applied to the input data [Fig. 1(c)]. Such structures mostly follow a cascade approach [4]–[6] where SVMs of increasing complexity are arranged in a hierarchy of stages. The SVM stages at the beginning of the hierarchy have lower computational complexity (i.e., need to process only a small number of SVs) and lower discrimination capabilities, and are tasked with removing the majority of samples from the negative class. The latter stages are then able to perform more accurate classification on the remaining samples. However, this incurs a higher computational cost as they need to process more SVs. Overall, an input sample needs to pass all stages to be classified as positive [Fig. 1(c)]. Under this scheme, a large amount of input samples are discarded early in the classification process by the stages at the beginning of the cascade, resulting in significant speedups. In addition, it is possible to use the reduced-set-method [13], to reduce the number of SVs required by the nonlinear kernel stages in order to further improve classification times. Furthermore, since the latter stages need to better discriminate between positive and negative samples, feature extraction algorithms may be used to improve accuracy, which, however, further increases computational demands.

C. Related Work

The speedups achieved by software implementations of cascade SVM classification schemes over monolithic, although significant, do not offer adequate performance for real-time resource-constrained applications [4]–[6], [15], [16]. This is because the latter stages become the bottleneck, since they require processing an increased number of SVs and the requirement for parallel processing arises. Hence, hardware acceleration [17], [18] of SVM classification has been explored in order to take advantage of the inherit parallelism of the SVM computation flow in an attempt to provide real-time and low-cost/low-power solutions.

The majority of the proposed hardware architectures attempt to improve the performance by employing parallel processing modules (PPMs), which process the elements of the input vector in parallel on FPGA platforms. However, for such architectures, the parallelism depends on the vector dimensionality for a given problem in terms of computational resources. When the vector dimensionality is high and the hardware resources for fully parallel processing are not available, the architecture can be folded to process the elements in groups. However, this increases the cycles needed to process a single vector. Hence, works that utilize such architectures have optimized it specifically for the vector dimensionality of the given problem and have been restricted to small scale data, with only a few hundred vectors and low dimensionality [9], [19], [20], and small-scale multiclass implementations [21] in order to be able to meet real-time constraints. In addition, these architectures cannot tradeoff processing more SVs rather than vector elements, and hence, cannot efficiently deal with the different computational demands of the cascade SVM stages.

Alternative approaches include FPGA coprocessors for parallel vector processing in order to speedup SVM computations [8], [22]. However, these architectures do not consider the kernel implementation and the FPGA is only used for the dot-product operations of the SVM classification flow. Furthermore, the parallel processing capabilities depend on parallel input through the Peripheral Component Interconnect express and external DRAM, which have high power consumption and are thus unsuitable for embedded applications. Another approach [23] is to dedicate a multiply accumulate unit per SV to process them in parallel with a single input vector. However, such architectures are limited by the number of SVs and also cannot be used to parallelize the processing of a linear SVM.

Research has also been conducted on potential simplifications to make the SVM classification more suitable for hardware implementation on devices with limited computational resources. These approaches include using CORDIC algorithms to compute the kernel functions [10], [19], [24], [25]. However, low-resource-consuming implementations of CORDIC algorithms have increased latency [10]. Khan *et al.* [26] and Boni and Zorat [27] propose that computations are done in the logarithmic number system, where multiplications are replaced with additions, in order to reduce the required processing resources. However, they only consider a single processing module, hence, when adopting a more parallel architecture, to facilitate real-time operation, the additional cost from converting between the decimal number system to the logarithmic one and back again for all inputs increases. Boni *et al.* [25], Anguita *et al.* [28], [29], and Ghio and Pischiutta [30] have looked at how the bitwidth precision impacts the classification

error, in an effort to find the best tradeoff between hardware resources, performance and classification rate. Although the kernel operations still need to be implemented with multipliers leading to high-resource demands for parallel implementations. A hardware friendly kernel was proposed in [31], which operates in conjunction with a CORDIC algorithm and addresses the resource requirements for SVM implementation. However, this kernel does not address the memory requirements of SVs. In contrast, our approach also reduces the memory demands for the storage of SVs and alpha coefficients. Furthermore, as previously mentioned CORDIC-like algorithms can have a negative impact for parallel implementations targeting high performance.

NVidia's Compute Unified Device Architecture has been used in [7], [32], and [33] in order to speedup SVM classification using the parallel computing resources of a GPU, showing improved results compared with CPU implementations. However, GPUs are power hungry devices compared with FPGAs [22], [34] (FPGAs consume approximately an order of magnitude less power, as shown in [11]) and as such they are not suitable for power-constrained embedded applications, such as image object classification. In addition, existing GPU implementations do not translate well to the more energy-efficient embedded GPUs due to less available resources (less memory, registers, cache, and cores) [35].

The above related works consider only monolithic SVM classifiers. Only recently, there has been some work in the hardware implementation of cascade SVM classifiers. Papadonikolakis and Bouganis [34] implement an architecture of cascade classifiers with low and high precision bitwidth and exploit the dynamic ranges of heterogeneous data set problems to achieve an efficient resource utilization. In contrast, in this paper, we exploit the characteristic of a specific class of problems where samples of one class appear more frequently than the other to design an optimized hardware architecture.

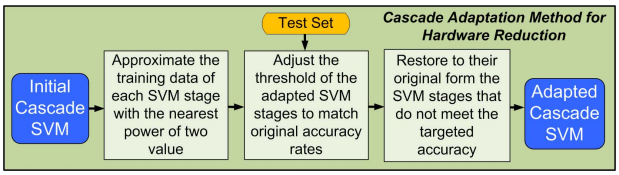
Summarizing, in their majority, most of the previously presented works are application specific, and efficient ways to utilize the different computational demands of cascade SVMs stages have not been sufficiently examined. Moving toward large-scale embedded applications and problems where thousands of samples need to be classified in real time, the majority of which belong to one of the two classes, cascade SVMs will need to be utilized to provide speedups. As such, single SVM architectures, which do not exploit the properties of the cascade classification scheme, are not suited for this purpose. Hence, this paper is one of the first to explore the potential of a flexible and parallel hardware architecture and design methods that can be used to improve different aspects of SVM hardware architectures.

III. Proposed Hardware Architecture and Methods

Cascade SVMs have demonstrated improvements over conventional SVM models (i.e., monolithic) in terms of classification speed [4]. However, it is still challenging to achieve real-time performance, especially as the amount of input samples that need to be classified increases. Hence, we propose a parallel hardware architecture to provide higher classification throughput and a hardware-reduction method leading to a more compact hardware implementation suitable for embedded system applications. In addition, this paper also develops a novel method to improve classification speed by taking advantage of cascade classification information to reduce the amount of input samples that reach the more computationally intensive latter cascade stages. Finally, in many classification problems, some form of feature extraction/preprocessing method needs to take place in order to deal with different variations and improve detection accuracy. Hence, the architecture also incorporates a feature extraction algorithm based on local binary pattern (LBP) descriptors, targeting object detection applications.

A. Cascade SVM Hardware-Reduction Method

The proposed hardware-reduction method exploits the fact that early stages in an SVM cascade are nonoptimal classifiers in order to reduce the resources needed for their hardware implementation, by adapting their parameters (SVs and alpha coefficients) while maintaining their ability to discard a large amount of negative samples. The proposed hardware-reduction method is to modify the SVs and alpha values of the low-complexity SVMs by rounding them to the nearest power of two values instead of using the conventional fixed-point representation approach. Consequently, all multiplication operations in the SVM classification phase (the kernel dot-product calculations and computations related to the alpha coefficients) will become shift operations, which require less resources to be implemented in hardware. In addition, since the SVs and alpha coefficients are now power of two values, there is no need to store the binary representations of decimal numbers but only shift data (shift amount, shift direction, and number sign). This will result in an adapted cascade SVM with reduced storage and computational demands. However, the expense from approximating the SVs and alpha coefficients with powers of two comes with the modified resulting classification accuracy will be different from that of the initial SVM cascade. The receiver-operating-characteristic (ROC) curve of each cascade stage whose parameters have been rounded off to the nearest power of two is used to adjust its accuracy to similar rates of that of the initial cascade stages. The ROC curve shows the performance of a binary classifier by illustrating the corresponding true positive (TP) and false positive (FP) rates, given a test set. As such, by setting the appropriate threshold, the performance of the adapted stages in the SVM cascade can be adjusted to match the TP rate of the initial SVM cascade stages. This is necessary, since we are interested in maintaining the TP rate. There are tradeoffs which stem from changing the original classification model. In particular, the reduced computational and storage requirements come at a cost of an increase in the FP rate of the adapted classifiers, as shown in Fig. 10. However, the overall accuracy tends to meet the accuracy of the final classification stage and hence the increase is not significant (only a 1% drop, as shown in Section IV). Adapted stages, which do not yield the targeted accuracy, are reverted back to the initial model. The process is shown in Fig. 2(a).



(a)

1. {**Input:** Set of positive and negative samples not used in cascade SVM training $(x_1,y_1)\ldots(x_n,y_n)$ where y_i = *{-1,1}*. Cascade SVM stages.}
2. **Initialization:** Select the desired first *m* SVM cascade stages
3. **For** i = 1,…,*n*
   For j = 1,…,*m*
   a. Classify sample x_i with *SVM_stage(j)*.
   b. Get *SVM_stage(j)* response → *SVM_out(i,j)*.
   c. If predicted "negative" move to the next sample.
   d. If predicted "positive" move to next stage.
4. Collect the response values of the samples that have been predicted as "positive" by all *m* stages to form a test set D → [*SVM_out(i,1…j) yi*].
5. Separate the data set into training D_{Tr} and test sets D_{Te}.
6. Train a machine learning algorithm using D_{Tr}.
7. Evaluate trained model accuracy using D_{Te}.
8. Adjust threshold to achieve the desired true positive accuracy rate using ROC curve.
9. {**Output:** Response Evaluation Machine Learning Algorithm}

(b)

Fig. 2. (a) Hardware-reduction method. (b) Response evaluation method.

The hardware-reduction process takes place after the cascade structure is decided, meaning that the kernel function, and a number of SVs or RSVs for each SVM cascade stage are determined. As such, the proposed method can easily be used with different SVM training frameworks. Furthermore, the method does not depend on the specific hardware architecture used for the implementation of the cascade and as such can be optimized to different architecture requirements.

B. Cascade Response Evaluation Method

Exploiting cascade information is a common technique used to speedup the training phase of cascade classifiers by eliminating samples from the training set. However, so far only a few works have attempted to do something similar in the classification phase. These methods [36] perform a joint logical operation (AND–OR) on the outcome of the cascade stages in order to correct/reevaluate the detection result. Such methods are usually used to improve detection and require that all the stages process the input data in order to reach a decision. However, this means that the overall detection speed is reduced. To improve performance, there is a need for a mechanism that can indicate whether an input sample needs to move on to the more computationally demanding stages. In this paper, we propose to do this by examining the responses of early cascade stages in order to rapidly eliminate data samples prior to reaching the latter stages. It is based on the observation that when looked at collectively, the responses of the individual cascade stages can exhibit patterns which can help in discriminating between samples belonging to different classes. This adds an additional dimension to the cascade classification phase that amongst others can be used to speedup the overall process.

Such a response processing mechanism can be constructed by following the process shown in Fig. 2(b). An integral part of this process is the construction of the training and test sets. Examples of positive and negative samples not used in the training phase of the SVM cascade are collected. Then, these are fed to the selected cascade stages in order to collect the response of each stage and construct a corresponding response feature vector. Next, we select the response vectors of the samples, which are predicted to belong to the positive class (i.e., have passed all stages) to form the set of response vectors, which will be used to construct the response classifier. We then separate this set into the training and test sets both of which must contain responses obtained from true negative and positive samples. Using this new training set, a machine learning algorithm, which will act as a response evaluator, can be trained to discriminate between different responses. Of course, the positive and negative samples can often have similar cascade responses. Hence, the training goal for the machine learning algorithm is to make sure that the positive responses will be correctly classified so that the TP accuracy of the whole cascade is not affected. With regard to the responses corresponding to negative classes, any correct classification is beneficial, since those samples will not need to be classified in the final stage. The desired TP rate can be adjusted experimentally by setting an appropriate threshold value. This is a general approach of handling the cascade responses and thus can be used similarly to benefit both software and hardware implementations. With regard to software implementations, the additional computations necessary for the latter cascade stages are eliminated, while for hardware implementations, the reduced workload can result in more compact architecture implementations for the latter stages. In this paper, we focus on the hardware aspects of this approach and the benefits of using this mechanism are outlined by the results in Section IV.

C. Hardware Architecture

The proposed architecture (Fig. 3) consists of cascade processing components as well as additional components, which relate to the targeted benchmark application of object detection, an embedded application where samples of one class (nonobject class) appear more frequently than the other (object class) [4]. The presented architecture is comprised of flexible and generic components and the parameters of each one can be adjusted to meet the given requirements, such as different data sizes and image dimensions, thus facilitating the design of an optimized hardware accelerator that is tailor-made for a specific application. Furthermore, the modular design means that the architecture can support different processing modules which allows it to implement the operations required by each SVM in the cascade.

1) Cascade SVM Hardware Architecture: The proposed hardware architecture considers the throughput and processing needs of each stage in the cascade. Accordingly, the proposed hardware architecture for the cascaded SVM classifier consists of two main processing modules, which provide different parallelism with respect to the input data and SVs in order to meet the different demands of the cascade SVM models, and also the amount of input data that each will need to process. The first is a PPM, which performs the processing necessary for all the adapted SVM stages (Fig. 4). The second is a sequential processing module (SPM), shown in Fig. 5, optimized for the high-complexity SVM stages. The cascade response processing is implemented with

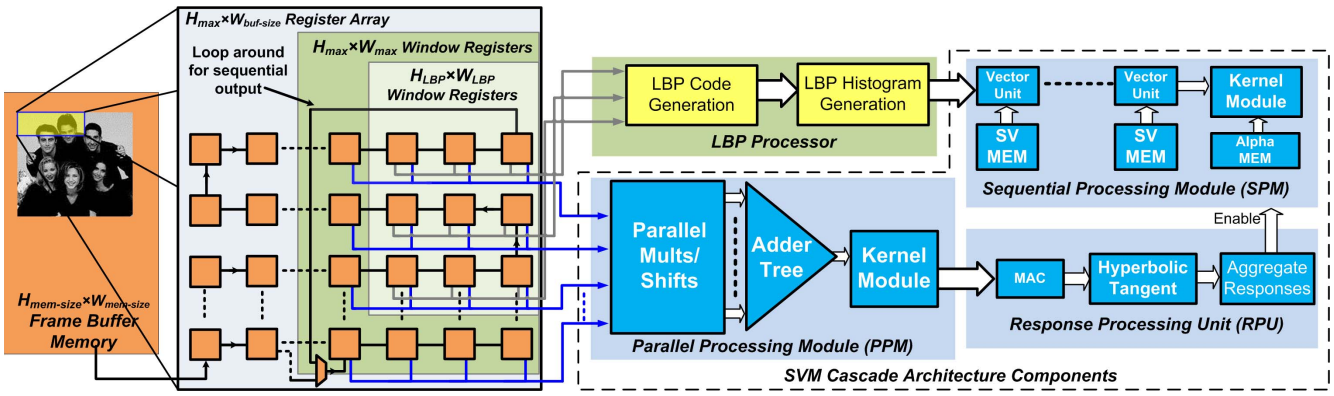


Fig. 3. SVM cascade system architecture comprised of the SPM, the PPM, the register array, frame buffer memory, the LBP processor, and the RPU.

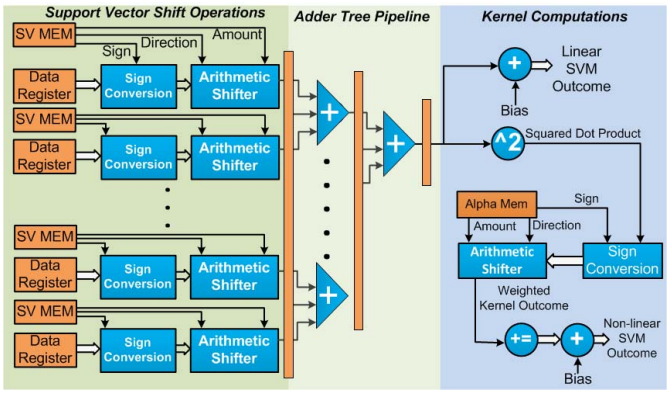


Fig. 4. PPM architecture.

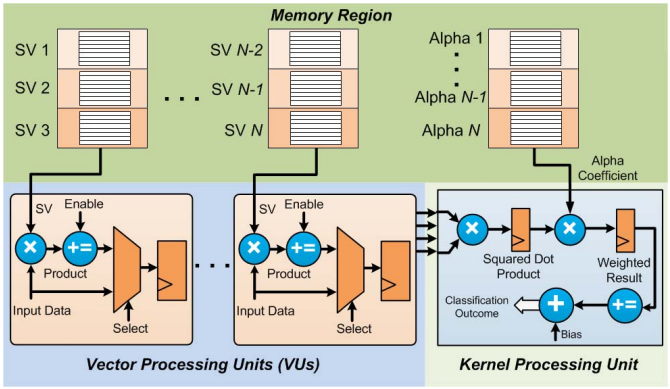


Fig. 5. SPM architecture.

a low-resource-consuming NN architecture to minimize hardware overheads while boosting performance.

a) Parallel processing module: The PPM handles the processing of the low-complexity SVM stages, which have been adapted using the proposed hardware-reduction method. In particular, the proposed architecture can process linear and second-degree polynomial kernels, but the plug-and-play approach of the architecture means that other kernel modules implementing different kernel functions can be used instead [37]. The characteristic of the early cascade stages is that they require processing of only a few SVs per input vector while having to process the majority of input vectors. As such, parallelism focuses on processing vector elements in parallel to reduce the processing time per vector.

The architecture of the PPM (Fig. 4) is comprised of three main regions: 1) SVM shift operations; 2) adder tree pipeline; and 3) kernel computation. The first region is comprised of parallel SV data memories, arithmetic shifters, and parallel sign conversion units. The second region is comprised of a tree of adders that sum the results of the previous stage in order to calculate the dot-product scalar value. The final region is dedicated to kernel processing and is also mostly implemented using arithmetic shift units. The operation of the PPM begins with the processing of the input vector elements by the sign conversion units, which are used to preserve the sign of the initial multiplication operation. The signed numbers are then processed by arithmetic shift units, which perform the shift according to the data that they receive from the memories. The shift data are fetched in parallel from small memory units, and include the sign of the SV, the shift amount, and the direction of the shift operation. The partial results are added together using a pipelined tree of adders so that the dot-product outcome can be obtained. The depth of the adder tree impacts the latency of the PPM and depends on the number of operands of individual adders used and the vector dimensionality, as well as the targeted frequency and amount of parallelism. The latency of the adder tree is given by

$$\text{adder\_tree\_stages} = \left\lceil \frac{\log(\text{vector\_dimensionality})}{\log(\text{adder\_input\_size})} \right\rceil. \quad (5)$$

Once the dot-product scalar value becomes available, the kernel computation follows. In the case of linear kernels (2), adding a bias value to the dot-product outcome will suffice in order to obtain the classification result. However, for second-degree polynomial kernels, as well as other kernels, the kernel computation module handles the latter steps of the classification phase. Only one multiplier is used in the PPM and is used to perform the square operation. The processing of the alpha coefficients is done with a sign conversion unit and an arithmetic shift unit similarly to the processing of the SVs. An accumulator is used to accumulate the result of each SV processing, and once all SVs are processed, an adder is used to process the bias with the accumulated result. The PPM stages

are pipelined, so one SV enters the pipeline every cycle. Hence, the total number of cycles needed to process the input vector at stage n is given by (6), where $N_{\mathrm{SV}}(i)$ is the number of SVs that need to be processed by stage i

$$\left(\sum_{i=1}^{n} N_{\mathrm{SV}}(i) + \text{adder\_tree\_stages} + 1\right). \tag{6}$$

The PPM architecture describes a fully unrolled implementation and allows for all vector elements to be processed in parallel, thus providing higher detection speeds. In cases where the resources are not available or the vector elements cannot be accessed in parallel due to limited I/O or memory access, the PPM architecture can be implemented using fewer resources by reducing the unrolling factor. Of course, this will have a negative impact on performance, which becomes more apparent as the number of SVs increases, as the time needed to process a single vector also increases.

b) Sequential processing module: The SPM is responsible for performing the computations necessary for the final SVM stage, which requires processing of hundreds of high-dimensional SVs. Hence, as the dimensionality of the vector increases, it becomes prohibitive in terms of resources and power to have multiple units in parallel for processing of a single vector, as the wiring and memory management complexities also increase. In addition, processing less vector elements while having to also process hundreds of SVs leads to a decreased performance. Hence, it is more efficient to use an alternative architecture, to that of the PPM, that will offer parallel processing tailored to the requirements of the more demanding SVMs [37]. In addition, since this module will be used less frequently, a flexible yet compact architecture is required.

This is achieved with the architecture shown in Fig. 5, which is comprised of a series of pipelined processing and memory elements [37]. The majority of the units in the module are vector processing units (VUs) and each unit handles the dot-product for one SV with the input vector. They are comprised of a multiply accumulate unit, and also a block RAM (BRAM), which acts as ROM and contains the data for one or more SVs, along with register and multiplexer logic for data transfer between vector units. The final unit in the pipeline is the kernel processing unit, which is equipped with multipliers and accumulators to carry out the scalar operations of the SVM processing flow. Multiple PPMs can be arranged in an array as in [37] to increase parallelism.

The input vector is processed with a group of SVs at a time, and each VU handles the processing of one SV. Once a group of SVs is processed, the next group follows. In total, depending on the number of groups, a total of $\lceil N_{\mathrm{SV}}/\text{num\_of\_VUs}\rceil$ processing repetitions are necessary. Hence, the size of the pipeline can be adjusted to fit the available resources and processing requirements by adjusting the number of SV groups. Each VU in the pipeline processes one SV with the input vector at a time. The data in the SPM flow in different directions through dedicated transfer mechanisms. The input vector values and VU results are propagated from the first unit to the next through

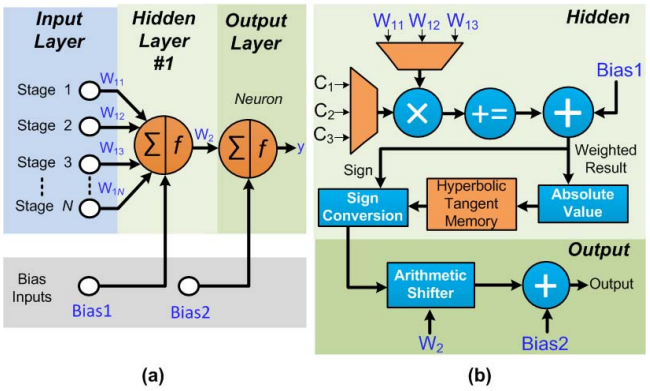


Fig. 6. RPU. (a) NN model. (b) NN-based RPU hardware architecture.

a register pipeline, while the SV data are fed to the VUs through parallel memories. When the processing of the input vector with the group of SVs is done, after vector_dim cycles, the multiplexers and registers in each vector unit are used to switch from propagating input vector values to scalar results. The scalar values are transferred sequentially through the pipeline and it takes num_of_VUs cycles for them to be processed by the kernel processing unit (with a two-cycle initial delay due to the pipeline stages). In this way, the kernel processing unit is shared between the units, reducing hardware requirements and also making it easy for the designer to substitute it with the desired kernel without having to change much of the system functionality. Each scalar value that enters the kernel unit is processed by the kernel operation and the alpha coefficient. In the case of the kernel described by (3), the operation involves a multiplier to find the square of the value and multiply accumulate units to process the alpha coefficients. Once all scalar values are processed, the final classification result is obtained by adding the bias to the accumulated result. Overall, the number of cycles needed to process an input vector is given by

$$\lceil N_{\mathrm{SV}}/\text{num\_of\_VUs}\rceil \times (\text{vector\_dim} + \text{num\_of\_VUs} + 2). \tag{7}$$

c) Response processing unit: As previously described, the objective of the cascade response evaluation process is to remove samples prior to the final SVM classification in order to improve processing speed. As such, it acts as a complementary stage to the overall cascade structure and can be used with any number of cascade stages. However, this needs to be done in a hardware-efficient manner in order to maintain performance and keep low-resource utilization. Hence, computationally and memory intensive algorithms are not the desired choice. For this reason, a computationally efficient feed-forward NN model is selected to perform the response evaluation process, as shown in Section IV, leads to a low-resource-consuming architecture that can sufficiently differentiate between responses.

The NN model, shown in Fig. 6(a), has a two layer structure with one neuron in each layer in order to keep the resource requirements low. The first neuron receives the responses from the cascade stages, multiplies them with their

respective weights, and accumulates the products. Then, it adds the bias value and sends it through a hyperbolic tangent activation function to the output neuron, which performs the same process and generates the classification outcome.

The NN hardware architecture [Fig. 6(b)] processes a different number of inputs depending on the number of cascade responses produced by the desired stages. Since each response is generated at different time intervals, it can be processed sequentially once it becomes available by the PPM. Multiplexers are utilized to select the output of the desired classifier and its corresponding weight value, which is represented in a fixed point format. The two values are multiplied and accumulated. Once all the cascade responses are accumulated the bias is processed. A lookup table (LUT) memory is used to implement the hyperbolic tangent function. We exploit the fact that this function is symmetric with respect to negative and positive inputs, and that its results range from $[-1, 1]$. Consequently, only the results for positive numbers are stored with the input being processed to obtain its absolute value. This leads to a more compact and efficient implementation. The sign of weighted accumulated sum is used to adjust the result of the hyperbolic function memory after the appropriate value is loaded, since it is the same for negative and positive values. Then, it is processed with the output layer weight, which is implemented using an arithmetic shift unit. Finally, the bias is added and the final outcome is computed. It is not necessary to use a hyperbolic function for the output layer neuron, since it does not change the sign of the result, which determines the class. The response processing unit (RPU) takes (NN_pipeline_stages + 2) cycles to process the response vector that is generated from the PPM.

d) Cascade processing flow: The architecture processes a single input vector at a time starting from the early stages implemented using the PPM. The RPU follows next to classify the responses of preceding stages if the input vector has been classified as positive. If the response evaluation predicts a positive sample, the RPU informs the SPM which in turn proceeds to classify that sample to obtain the final classification result. The different throughput requirements of the cascade SVM processing modules require an I/O mechanism that can adjust for parallel as well as sequential data transfer depending on the needs of each module. It should also take advantage of the application-specific characteristics to facilitate data reuse and reduce memory accesses. Furthermore, the cascade I/O structure should be able to handle classifier demands for different data points and data access patterns. Such architecture can be designed using a register array (Fig. 3) where data can be loaded to the array and outputted in parallel for the PPM and sequentially for the SPM.

2) I/O and Preprocessing for Object Detection: Additional components are incorporated into the architecture in order to handle the data flow and preprocessing for object detection, which requires processing data from the input image in a sliding window fashion to classify them as object or not. As such, the register array structure (Fig. 3) is also optimized for the object detection data flow so that it does not only provide sequential and parallel data access to the two processing modules but also to take advantage of potential data

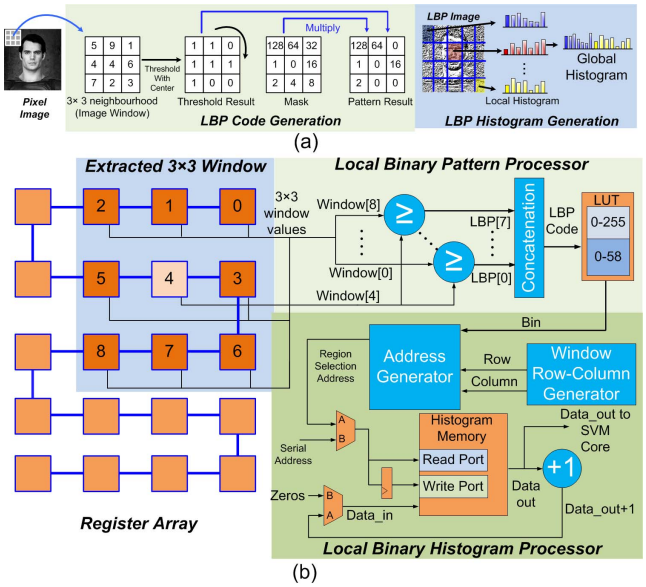


Fig. 7. (a) LBP descriptor. (b) LBP processing unit architecture.

overlap and reduce memory I/O. A frame buffer is employed to hold part of the image for fast local access. Finally, the architecture incorporates a specialized processor that performs LBP histogram extraction, which is used as features for object detection classification.

a) Object detection processing flow and I/O: An optimized I/O mechanism for object detection can be developed based on register array structure (Fig. 3) that provides different access patterns and window data selection for the image segment that is currently being processed. The register array has a size of size $H_{\max} \times W_{\text{buf\_size}}$, where $H_{\max}$ is the height of the window, and $W_{\text{buf\_size}}$ corresponds to the width of the array (i.e., how may additional image columns are stored). The input image pixels enter the register array and are propagated rowwise into the structure. The image region that resides at the right-most part of the register array corresponds to a single $H_{\max} \times W_{\max}$ window, which is the active window that feeds the processing units with data. In this data flow, the image region is processed in a window-by-window fashion. Once a window has been processed, a part of it is shifted out of the array, while new pixels are shifted in. Thus, a new window is formed at the rightmost region of the scanline buffer and is ready to be processed next. The data flow of the rightmost registers changes depending on whether the data are used for parallel or sequential processing. In the case of the PPM, window data are outputted and processed in parallel. In the case of sequential processing, which happens when the LBP features are generated, the registers form a chain so that data are outputted sequentially.

b) Local binary pattern processing unit: LBPs describe the relationship between a pixel and its neighborhood, and have been used in a wide range of computer vision applications [38]. Their major advantage is their low computational complexity [39], which makes them suitable for embedded applications. Generation of the LBP descriptor [38] consists of the following steps [Fig. 7(a)].

1) Compare the values in a 3×3 neighborhood against a threshold (the center pixel or the window mean value) placing 1 where the value is greater or equal, and 0 otherwise.
2) Multiply the resulting binary map with powers of two mask.
3) Sum the values to obtain the LBP code.
4) Divide the LBP-based image into k blocks of $i \times j$ pixels (e.g., 4×4 and 8×8) and construct a local histogram of l bins for each block.
5) Concatenate the local histograms to form a single global histogram descriptor.

The LBP descriptors can be used as features by the latter SVM stages, which require better discrimination capabilities. Since only a fraction of input data will be processed using LBP, it is more efficient to explore a low area overhead architecture.

Accordingly, the developed LBP processor architecture, shown in Fig. 7(b), processes a single 3×3 image neighborhood from the input image at a time, to reduce processing requirements. It receives the values of that window in parallel every cycle from the register array structure. Each window value is compared against the center window value in parallel through dedicated comparators and the results are concatenated to generate the LBP code. The number of transitions in the LBP code is found next in order to identify it as uniform (which has two or less transitions, e.g., 11110000) or nonuniform (which has more than two transitions, e.g., 10100101) [38]. The local histogram computation, which counts the uniform LBP codes against the nonuniform, follows next for each block in the LBP image. Since the bin of each LBP code is predetermined [38], an LUT is used to map the code to one of 59 possible histogram bins. Multiple local histograms are stored in the same central memory (of size $k \times l$), hence the hardware architecture needs to know the position of each LBP code in the image in order to determine the local histogram it belongs to. This is achieved by counting the row and column of each LBP code and monitoring the most significant bits of the row and column coordinates to indicate its corresponding block. Then, by setting the appropriate address offset the corresponding local histogram region is selected and updated. A dual-ported memory is utilized to facilitate fast update and reset of the histogram values. The updated histogram bin is read from one port and written to the other in the following cycle so no delays are observed. Second, this allows for an immediate reset to be performed right after the value is read from the SVM classification module to prepare the memory for the next histogram.

IV. Experimental Platform and Results

The proposed hardware architecture and methods were evaluated using the popular embedded application of face detection, which has also been used by software implementations of cascade SVMs. The cascade structure was trained using MATLAB and was used for the evaluation of the architecture and proposed methods, on 800×600 [Super Video Graphics Array (SVGA)] resolution images, in terms of frame rate, detection accuracy, power consumption,

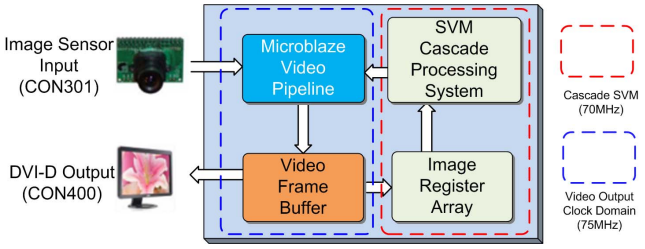


Fig. 8. Block diagram of the FPGA system.

TABLE I
Cascade Detection System Parameters

Search Window Size	20×20 ($H_{max} \times W_{max}$)	LBP Block Size	$i = 3$, $j = 6$
Downsampling Rate	1.2 (18 scales)	Number of LBP Blocks	$k = 18$
Window Step	5 pixels	LBP Histogram Bins	$l = 59$
Image Resolution	800×600 (SVGA)	Number of Windows	56984

as well as requirements in terms of computing resources. In addition, the proposed hardware architecture, which will be referred to as the adapted cascade, is compared against a baseline system, which implements the same cascade SVM structure, including the RPU (Stage 4), but without applying the hardware-reduction method, and thus the PPM is implemented using multipliers. Both implementations were evaluated and compared using a Xilinx Spartan-6 Industrial Video Processing board equipped with a Spartan-6 XC6SLX150T FPGA (Fig. 8). A Microblaze-based system was used for I/O and verification purposes, while for both systems, an on-chip buffer is used to store the input image and a register array for data loading and processing, which was experimentally found to provide an adequate balancing between I/O delays and hardware resources. Sections IV-A–IV-D detail the evaluation process and the results, while Section E shows comparison with related work.

A. SVM Cascade Training and Accuracy

The training of the SVMs and NN was conducted offline using MATLAB with kernels and parameters similar to what has been used in [4]–[6]. The resulting classification models were used to evaluate the proposed hardware architecture and approaches for online classification on an FPGA. We performed experiments with all possible LBP histogram parameters and selected those which provided the best accuracy results. The used parameter values are shown in Table I.

Positive and negative samples from [40] were used to set up an initial training set, which was later enhanced with additional samples. The first three cascade stages were trained in an incremental fashion [4], [6], [15]. The final SVM stage was excluded from the process and was trained using the complete training set, which was first processed using the LBP feature extraction. The first polynomial SVM (Stage 3, Fig. 9) was reduced to 20 RSVs, which was the smaller number of reduced vectors needed to maintain the original accuracy. In contrast, 100 RSVs were needed to maintain the accuracy [6], [15], for the final stage (Stage 5, Fig. 9). The first three stages

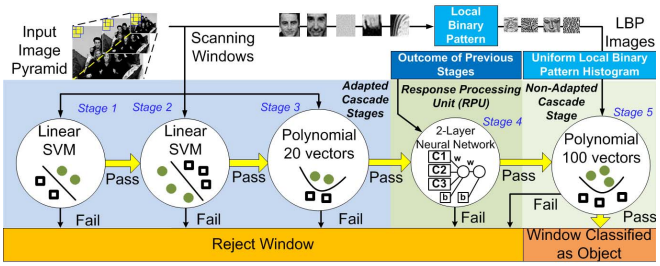
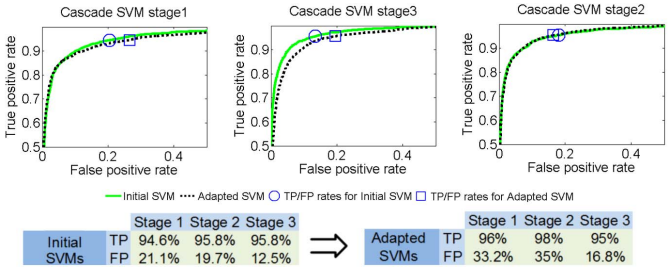


Fig. 9. Cascade SVM structure.



		Stage 1	Stage 2	Stage 3
Initial SVMs	TP	94.6%	95.8%	95.8%
	FP	21.1%	19.7%	12.5%
Adapted SVMs	TP	96%	98%	95%
	FP	33.2%	35%	16.8%

Fig. 10. Restore accuracy using the ROC curves. Top: part of ROC curves. Bottom: accuracies and error after adaptation.

retained a similar accuracy level after being rounded-off to the nearest power of two, as shown in Fig. 10, and hence were implemented on the PPM. However, for the final stage, there was a significant discrepancy between the classification accuracies of the adapted and original models. Hence, it was not approximated and the original model was used.

After the SVM cascade, the training of the NN-based RPU followed using the process described in Fig. 2. The feed-forward NN consisted of one neuron for each of the two layers and was trained using the gradient descent with momentum and learning rate backpropagation algorithm in MATLAB. For this purpose, 30 515 positive and 329 383 negative window samples, not used in the SVM training phase, where extracted from various images and were passed through the first three adapted SVM cascade stages to collect their responses. This resulted in a 3-D response vector per sample. The response vectors of the samples classified as positive by the cascade (which also include truly negative samples) were selected to form a new set (29 117 response vectors for positive samples and 8803 response vectors for negative samples). This new pool of response vectors was then partitioned in a training and test set, both containing responses from negative and positive samples, in order to train and evaluate the NN-based Response Processing Unit respectively. A subset of cascade responses for the training set is shown in Fig. 11, where it is evident that the responses of the early stages exhibit different patterns for positive and negative class samples. The NN-based RPU training resulted in a correct classification rate of 99% for positive and 60% for negative cascade responses using the constructed responses test set.

B. FPGA Implementation and Logic Resource Utilization

The two cascade implementations (baseline and adapted) have the same basic architecture (Fig. 3) and data flow.

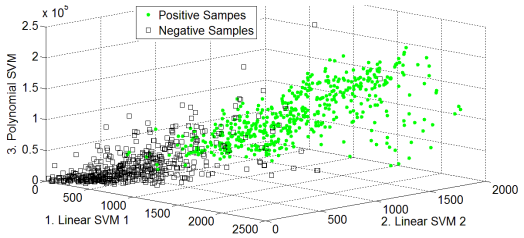


Fig. 11. Response vectors produced by the first three SVM cascade stages for negative (square) and positive (filled circle) samples.

TABLE II
FPGA RESOURCE REQUIREMENTS PER UNIT AND SYSTEM

FPGA Resources	Registers (184304)	LUTs (92152)	BRAMs (268)	DSPs (180)
SPM	1736 (1%)	2241 (2%)	51(19%)	50 (27%)
Adapted PPM	2679 (1%)	19006 (20%)	1 (<1%)	---
Baseline PPM	3724 (2%)	30791 (33%)		
NN-based RPU	82 (<1%)	379 (<1%)	2 (<1%)	6 (3%)
LBP Processor	32 (<1%)	94 (<1%)	2 (<1%)	---
Memory & I/O Units	1831 (1%)	1200 (1%)	180 (67%)	---
Microblaze Video Pipeline	10780 (5%)	9891 (10%)	20 (7%)	3 (2%)
Baseline Cascade System	21214 (11%)	47396 (51%)	256 (96%)	59 (32%)
Adapted Cascade System	20153 (11%)	35532 (38%)		

The PPM architecture was based on a fully unrolled implementation, while the SPM was implemented with 50 DSP units (num_of_VUs = 50) meaning that the input data to the SPM are processed two times with different SV groups. The NN-based RPU was mapped on the FPGA LUTs with a BRAM used for the hyperbolic tangent implementation. The only difference between the two implementations is that in the adapted cascade case the PPM was optimized using the hardware-reduction method from Section III-A. Consequently, the multiplication units were replaced with shift units and the SV data stored in the training data ROMs corresponded to shift values instead of real number values. Each ROM holds the SV data for the first three cascade SVM stages for the specific vector elements. Finally, a single BRAM was used to implement the hyperbolic tangent function of the NN-based RPU. Both implementations on the Xilinx Spartan-6 XC6SLX150T FPGA have the same critical path, the SPM kernel unit mapped on the DSPs, and as such have the same operating frequency of 70 MHz. The implementation of the adapted PPM requires 40% fewer FPGA logic resources compared with the baseline PPM. This is reflected with a 25% reduction in the utilized resources when considering full system implementations, as shown in Table II. Overall, the proposed approaches can be used to meet different constraints in the design space for different FPGAs from low-end to high-end. For low-end devices, the immediate impact is a method to better fit the design to limited resources, while for high-end with enough resources, power consumption can be reduced by

Fig. 12. Detection results on CMU-MIT images.

TABLE III
STATISTICS FOR EACH CASCADE STAGE

Cascade Stages	Stage 1 (PPM)	Stage 2 (PPM)	Stage 3 (PPM)	Stage 4 (RPU)	Stage 5 (LBP & SPM)
Windows Processed	56984 (100%)	3025 (5%)	2334 (4%)	713 (1,2%)	228 (0,4%)
Rejection Rate	94,6%	22,8%	69,4%	76,4%	---
Cumulative Cycles	9	10	30	35	2697
Vectors per stage $N_{SV}(i)$	1	1	20	---	100

changing the multiplication units to shift units. Furthermore, for both cases, the architecture components can be optimized to meet the available FPGAs resources.

C. Classification System Accuracy and Frame Rate

This section outlines results related to accuracy and frame rate, two important metrics in object detection, and also highlights the overall impact of the LBP processor and RPU. The accuracy of the adapted cascade SVM was evaluated on the widely used CMU-MIT database of faces [41]. In addition, images from the data set were cropped and resized to 800×600 (SVGA) resolution and used to evaluate the frame rate of the cascade SVM implementation. Some full frame detection results are shown in Fig. 12. Each 800×600 image generates a total of 56 984 20×20 search windows for 18 scales and a window step of 5 pixels. Each frame requires a different time to be processed, by the cascade implementations, depending on how many windows reach each stage, and by how many cycles it takes a stage to process an input. All windows are processed by the first SVM stage, however, only $\sim$1% of them reach the final SVM stage, as shown in Table III. In addition to the actual processing time, the I/O delays per frame also negatively impact classification speed. To achieve higher detection rates, I/O and memory operations overlap with processing.

As shown in Fig. 10, the adapted cascade SVM stages have similar accuracy to that of the initial SVMs in terms of TP detection accuracy. However, the FP rate has increased between 4% and 15%. This is to be expected, since the approximations introduced a discrepancy between the initial and adapted SVM models. However, the final detection accuracy (Fig. 13) of the adapted cascade is determined by the latter stages, and so any discrepancies are effectively masked. Hence, both implementations are expected to have a similar overall accuracy ($\sim$80%). For more details on the effects of the approximations on the accuracy, we direct the readers to [12].

Results for different system configurations of the adapted SVM cascade, with and without the LBP and RPU,

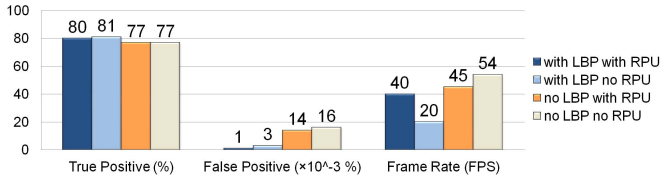


Fig. 13. Comparative results for different configurations.

are shown in Fig. 13. The cascade SVM boosted by the NN-based RPU was able to achieve an accuracy of 80%, which was only 1% less than the same system without the RPU, which suggests that with additional training and enhancement of the data sets, it would be possible to achieve the same accuracy. Nevertheless, the minimal drop in accuracy, when using the RPU, is offset by a $2\times$ increase in performance. It allows the cascade system to operate at $\sim$40 frames/s instead of $\sim$20 frames/s, making the system capable of real-time operation. This happens because even though most windows are discarded by the first two cascade stages, the NN-based RPU manages to reduce the number of windows ($\sim$230 instead of $\sim$715, Table III) that reach the slower SPM. Furthermore, the introduction of the LBP feature extraction process helped improve both the TP rate as well as the FP rate, the latter by an order of magnitude. In addition, since the LBP features are only extracted during the final stage the improved accuracy has only a small impact on the frame rate (40 instead of 45 frames/s). The results also indicate that in cases where the frame rate is of much higher importance than accuracy (e.g., when processing videos from a static environment), the optimized SVM cascade without the LBP and RPU can also be used to offer higher performance. Overall, through the use of LBP features to improve accuracy, and the RPU to boost the frame rate, we achieve an adequate tradeoff between frame rate and detection accuracy to meet application requirements.

D. Power Consumption

Power analysis tools from Xilinx were used to measure power consumption demands of the adapted and baseline cascade SVM FPGA implementations. The characteristic of the cascade architectures is that the PPM and SPM are not used at the same time, since they implement different cascade stages. Hence, the dynamic power consumption ranges depending on which module is active. The total power budget, including the Microblaze I/O system, for the adapted cascade SVM system ranges from 4.1 to 8 W, while for the baseline cascade system, it ranges from 4.1 to 9.9 W. These figures correspond to the worst case scenario where all signals change every cycle. However, it is anticipated that on average the power consumption will be lower. The peak power consumption happens when the PPM module is used. The lowest consumption happens when the NN-based RPU is used, and when the SPM and LBP cores are used, power consumption reaches 4.9 W. Overall, the utilization of less LUT resources by the adapted PPM results in reducing the peak power needed for the adapted cascade system by $\sim$20%.

TABLE IV
COMPARISON OF RELATED WORKS FOR SVM FPGA-BASED HARDWARE IMPLEMENTATIONS OF OBJECT DETECTION SYSTEMS

Related Works		Rojas [23] [a]	Kryjak [39]	Bauer [33] [b]	Presented Work
Application		Barcode Detection	Head-Shoulder Detection	Pedestrian Detection	Face Detection
Method		Polynomial SVM, 88 SVs	Linear SVM & LBP	SVM (GPU) & HOG (FPGA)	Cascade SVM & LBP
Platform		Xilinx Virtex II Pro XCV3000	Xilinx Virtex 6 XC6VLX240T	Xilinx Spartan 3/NVIDIA GPU	Xilinx Spartan 6 XC6SLX150T
FPGA Resource	LUT	22938/28672	12068/150720	28616/62208 [c]	35532/92152
	REG	N/P	15893/301440	N/P [c]	20153/184304
	BRAM	160KB	124/416	100 [c]	256/268
	DSP	N/P	66/768	18/96 [c]	59/180
Image Size		512×512	640×480	800×600	800×600
Search Window Size		16×16	32×24	48×96	20×20
Feature Vector Size		256	1440	1980	400 & 1062
Number of SVs		88	1	N/P	122
Frequency		166 MHz	120 MHz	63 MHz [c]	70 MHz
Detection Accuracy		TP: 91.8% | FP: 4.2%	TP: 83%	TP: 95.4% | FP: 0.1%	TP ~80% | FP: ~0.001%
Detection Speed		N/P [a]	60 FPS	10 FPS	40 FPS

[a] Performance is 352 cycles per sample just for the vector operations. No I/O delays are included.
[b] A hybrid system where the GPU implements the SVM and the feature extraction based on HOG is implemented on the FPGA.
[c] These correspond only to the HOG implementation on the FPGA.
N/P – Not Provided | TP - True Positive | FP - False Positive

E. Related Work Comparison

Related works for object detection applications are shown in Table IV along with information regarding parameters and performance. These works use different algorithms, training and test sets, and benchmark applications, and so it is difficult to make a direct comparison between implementations. However, since the SVM classification flow treats all data as vectors, the number of samples and SVs processed, along with vector dimensionality, can provide an indication to the processing performance for each work.

A head–shoulder detection system is presented in [39]. It utilizes linear SVM and LBP descriptors to classify 19 200 windows from 640 × 480 images from an already known environment. It sacrifices accuracy for performance using a single linear SVM (with a clock frequency of 120 MHz) and processes only a few elements of the SV feature vector in parallel to keep the resource utilization low. In addition, foreground detection is used to compensate for the linear SVM. The implementation in [23] scans a 512×512 image in nonoverlapping blocks to perform bar-code detection. It performs the dot-product operations in 352 cycles for one window. However, the scalar operations are not included. It processes around 1024 16 × 16 window samples, corresponding to 256-D vectors, per image, without downscaling the input image, which simplifies the I/O and memory accesses. The hybrid FPGA-GPU pedestrian detection system [33] for 800 × 600 images is able to classify around 1000 windows. The lower throughput can be attributed to the larger feature size. However, the number of processed windows is an order of magnitude less than that in our work. In addition, the use of GPU may prohibit such implementations to be used in embedded applications due to power consumption constraints. Overall, in order to achieve real-time performance existing works rely on processing a few window samples, smaller image resolutions, or process a few SVs. Through the proposed architecture and methods, it is possible to process higher resolution images that generate more windows, with a higher number of SVs in real time (∼56 000/frame for ∼40 frames/s) while also reducing the implementation requirements.

The SVM hardware implementations target different applications and thus accuracy is difficult to compare directly. On the other hand, software-based implementations [4]–[6] have utilized cascade SVMs for face detection with accuracies that range between 78% and 80% with similar training set sizes and cascade structure to our work. The proposed optimized SVM cascade system achieves a detection rate of 80%, which is on par with these works while processing higher resolution images in real time.

V. CONCLUSION

The work presented in this paper considers the efficient hardware implementation of cascade SVMs, which can be used to design intelligent embedded systems for online real-time classification applications. The hybrid processing architecture takes advantage of the nature of the cascade classification structure and in conjunction with the hardware-reduction method and the novel response evaluation method, it manages to achieve adequate tradeoff between accuracy, performance, power, and resource utilization. The proposed architecture and methods can be used to design low-cost parallel SVM coprocessors to accelerate more demanding monolithic SVM classifiers or to optimize cascade SVM classifiers for embedded classification applications, thus allowing SVM architectures to tackle larger scale problems (e.g., classification on higher resolution images) to what has been addressed in the literature.

REFERENCES

[1] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995.
[2] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining Knowl. Discovery*, vol. 2, no. 2, pp. 121–167, 1998.
[3] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: An application to face detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 1997, pp. 130–136.

[4] B. Heisele, T. Serre, S. Prentice, and T. Poggio, "Hierarchical classification and feature reduction for fast face detection with support vector machines," *Pattern Recognit.*, vol. 36, no. 9, pp. 2007–2017, 2003.
[5] I. Kukenys and B. McCane, "Classifier cascades for support vector machines," in *Proc. 23rd Int. Conf. Image Vis. Comput.*, Nov. 2008, pp. 1–6.
[6] Y. Ma and X. Ding, "Face detection based on cost-sensitive support vector machines," in *Proc. 1st Int. Workshop Pattern Recognit. Support Vector Mach.*, 2002, pp. 260–267.
[7] B. Catanzaro, N. Sundaram, and K. Keutzer, "Fast support vector machine training and classification on graphics processors," in *Proc. 25th Int. Conf. Mach. Learn.*, 2009, pp. 104–111.
[8] S. Cadambi *et al.*, "A massively parallel FPGA-based coprocessor for support vector machines," in *Proc. 17th IEEE Int. Symp. Field Program. Custom Comput. Mach. (FCCM)*, Apr. 2009, pp. 115–122.
[9] O. Piña-Ramírez, R. Valdés-Cristerna, and O. Yáñez-Suárez, "An FPGA implementation of linear kernel support vector machines," in *Proc. IEEE Int. Conf. Reconfigurable Comput. FPGA's*, Sep. 2006, pp. 1–6.
[10] M. Ruiz-Llata, G. Guarnizo, and M. Yébenes-Calvino, "FPGA implementation of a support vector machine for classification and regression," in *Proc. Int. Joint Conf. Neural Netw.*, 2010, pp. 1–5.
[11] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2012, pp. 47–56.
[12] C. Kyrkou, T. Theocharides, and C.-S. Bouganis, "An embedded hardware-efficient architecture for real-time cascade support vector machine classification," in *Proc. 13th Int. Conf. Embedded Comput. Syst., Archit., Modeling, Simulation (SAMOS)*, Samos, Greece, Jul. 2013, pp. 129–136.
[13] C. J. C. Burges, "Simplified support vector decision rules," in *Proc. Int. Conf. Mach. Learn.*, 1996, pp. 71–77.
[14] H. Sahbi, D. Geman, and N. Boujemaa, "Face detection using coarse-to-fine support vector classifiers," in *Proc. Int. Conf. Image Process.*, Jun. 2002, pp. 925–928.
[15] S. Romdhani, P. Torr, B. Schölkopf, and A. Blake, "Efficient face detection by a cascaded support–vector machine expansion," *Proc. Roy. Soc. London A, Math., Phys. Eng. Sci.*, vol. 460, no. 2051, pp. 3283–3297, Nov. 2004.
[16] H.-X. Zhao and F. Magoulès, "Parallel support vector machines on multi-core and multiprocessor systems," in *Proc. 11th Int. Conf. Artif. Intell. Appl.*, 2011, pp. 75–81.
[17] 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.
[18] R. Genov and G. Cauwenberghs, "Kerneltron: Support vector 'machine' in silicon," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1426–1434, Sep. 2003.
[19] D. Mahmoodi, A. Soleimani, H. Khosravi, and M. Taghizadeh, "FPGA simulation of linear and nonlinear support vector machine," *J. Softw. Eng. Appl.*, vol. 4, no. 5, pp. 320–328, May 2011.
[20] I. Biasi, A. Boni, and A. Zorat, "A reconfigurable parallel architecture for SVM classification," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, Jul./Aug. 2005, pp. 2867–2872.
[21] T. Groleat, M. Arzel, and S. Vaton, "Hardware acceleration of SVM-based traffic classification on FPGA," in *Proc. 8th Int. Conf. Wireless Commun. Mobile Comput.*, Aug. 2012, pp. 443–449.
[22] H. P. Graf *et al.*, "A massively parallel digital learning processor," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 2008, pp. 529–536.
[23] R. Reyna-Rojas, D. Houzet, D. Dragomirescu, F. Carlier, and S. Ouadjaout, "Object recognition system-on-chip using the support vector machines," *EURASIP J. Adv. Signal Process.*, vol. 2005, pp. 993–1004, Jan. 2005.
[24] M. Ruiz-Llata and M. Yébenes-Calvino, "FPGA implementation of support vector machines for 3D object identification," in *Proc. 19th Int. Conf. Artif. Neural Netw.*, 2009, pp. 467–474.
[25] A. Boni, F. Pianegiani, and D. Petri, "Low-power and low-cost implementation of SVMs for smart sensors," *IEEE Trans. Instrum. Meas.*, vol. 56, no. 1, pp. 39–44, Feb. 2007.
[26] F. M. Khan, M. G. Arnold, and W. M. Pottenger, "Hardware-based support vector machine classification in logarithmic number systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2005, pp. 5154–5157.
[27] A. Boni and A. Zorat, "FPGA implementation of support vector machines with pseudo-logarithmic number representation," in *Proc. Int. Joint Conf. Neural Netw.*, 2006, pp. 618–624.
[28] D. Anguita, A. Ghio, S. Pischiutta, and S. Ridella, "A hardware-friendly support vector machine for embedded automotive applications," in *Proc. Int. Joint Conf. Neural Netw.*, 2007, pp. 1360–1364.
[29] D. Anguita, A. Ghio, and S. Pischiutta, "A learning machine for resource-limited adaptive hardware," in *Proc. 2nd NASA/ESA Conf. Adapt. Hardw. Syst.*, 2007, pp. 571–576.
[30] A. Ghio and S. Pischiutta, "A support vector machine based pedestrian recognition system on resource-limited hardware architectures," in *Proc. Res. Microelectron. Electron. Conf.*, 2007, pp. 161–163.
[31] D. Anguita, S. Pischiutta, S. Ridella, and D. Sterpi, "Feed-forward support vector machine without multipliers," *IEEE Trans. Neural Netw.*, vol. 17, no. 5, pp. 1328–1331, Sep. 2006.
[32] A. Carpenter. (2009). *CUSVM: A CUDA Implementation of Support Vector Machines*. [Online]. Available: http://patternsonascreen.net/cuSVM.html
[33] S. Bauer, S. Kohler, K. Doll, and U. Brunsmann, "FPGA-GPU architecture for kernel SVM pedestrian detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Workshops*, Jun. 2010, pp. 61–68.
[34] M. Papadonikolakis and C.-S. Bouganis, "Novel cascade FPGA accelerator for support vector machines classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 7, pp. 1040–1052, Jul. 2012.
[35] A. Maghazeh, U. D. Bordoloi, P. Eles, and Z. Peng, "General purpose computing on low-power embedded GPUs: Has it come of age?" in *Proc. 13th Int. Conf. Embedded Comput. Syst., Archit., Modeling, Simulation (SAMOS)*, Samos, Greece, Jul. 2013, pp. 1–10.
[36] M. M. Dundar and J. Bi, "Joint optimization of cascaded classifiers for computer aided detection," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2007, pp. 1–8.
[37] C. Kyrkou and T. Theocharides, "A parallel hardware architecture for real-time object detection with support vector machines," *IEEE Trans. Comput.*, vol. 61, no. 6, pp. 831–842, Jun. 2012.
[38] M. Pietikäinen, A. Hadid, G. Zhao, and T. Ahonen, *Computer Vision Using Local Binary Patterns*. London, U.K.: Springer-Verlag, 2011.
[39] T. Kryjak, M. Komorkiewicz, and M. Gorgon, "FPGA implementation of real-time head-shoulder detection using local binary patterns, SVM and foreground object detection," in *Proc. Int. Conf. Design Archit. Signal Image Process.*, 2012, pp. 1–8.
[40] *CBCL Face Database #1, MIT Center for Biological and Computation Learning*. [Online]. Available: http://cbcl.mit.edu/software-datasets/FaceData2.html, accessed Nov. 06, 2008.
[41] *CMU and MIT Face Database*. [Online]. Available: http://vasc.ri.cmu.edu/idb/html/face/frontal_images/, Nov. 6, 2008.

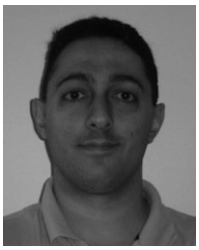
Christos Kyrkou (S'09–M'14) received the B.Sc., M.Sc., and Ph.D. degrees in computer engineering from the University of Cyprus, Nicosia, Cyprus, in 2008, 2010, and 2014, respectively.

He participated in various projects funded by the European Commission and the Research Promotion Foundation of Cyprus. He has been a Post-Doctoral Research Fellow with the KIOS Research Center for Intelligent Systems and Networks, University of Cyprus, since 2014. His current research interests include embedded systems, application-specific digital hardware accelerators, field programmable gate arrays, and reconfigurable hardware, computer arithmetic, computer vision, and pattern recognition.

Dr. Kyrkou is a member of the Association for Computing Machinery and the Technical Chamber of Cyprus. He received the award for graduating top of his class during his B.Sc. studies at the University of Cyprus, and the Full Scholarship for his post-graduate studies from the Department of Electrical and Computer Engineering, University of Cyprus.

Christos-Savvas Bouganis (S'01–M'03) is currently a Senior Lecturer with the Department of Electrical and Electronic Engineering, Imperial College London, London, U.K. He has authored over 40 research papers in peer-reviewed journals and international conferences, and 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.

Dr. Bouganis is a member of the Association for Computing Machinery. He serves on the program committees of many international conferences, including the Conference on Field Programmable Logic and Applications, the Conference on Field-Programmable Technology, the Design Automation and Test in Europe Conference, the Conference on the Signal Processing, Pattern Recognition, and Applications, and the Conference on Very Large Scale Integration. He is an Editorial Board Member of the IEEE TRANSACTIONS ON IMAGE PROCESSING, the *IET Computers and Digital Techniques*, and the *Journal of Systems Architecture*. He served as the General Chair of the Academic Resource Conference in 2008, and the Program Chair of the IET FPGA Designers' Forum in 2007.



Theocharis Theocharides (S'01–M'05–SM'11) is currently an Assistant Professor with the Department of Electrical and Computer Engineering and the KIOS Research Center for Intelligent Systems and Networks, University of Cyprus, Nicosia, Cyprus. His current research interests include intelligent embedded systems design, with a focus on the design of reliable and low power embedded and application-specific processors, media processors, and real-time digital artificial intelligence applications.

Dr. Theocharides serves on the Editorial Board of the *IEEE Design and Test Magazine* and on the TPC of several IEEE-sponsored and co-sponsored conferences.

Marios M. Polycarpou (M'92–SM'98–F'06) received the bachelor's degrees in computer science and electrical engineering from Rice University, Houston, TX, USA, in 1987, and the M.S. and Ph.D. degrees in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 1989 and 1992, respectively.

He is currently a Professor of Electrical and Computer Engineering and the Director of the KIOS Research Center for Intelligent Systems and Networks with the University of Cyprus, Nicosia, Cyprus. He has authored over 270 articles in refereed journals, edited books, and refereed conference proceedings, and co-authored seven books. He holds six patents. His current research interests include intelligent systems and networks, adaptive and cooperative control systems, computational intelligence, fault diagnosis, and distributed agents.

Prof. Polycarpou served as the President of the IEEE Computational Intelligence Society from 2012 to 2013. He served as the Editor-in-Chief of the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS from 2004 to 2010. He has participated in more than 60 research projects/grants funded by several agencies and industry in Europe and USA, including the prestigious European Research Council Advanced Grant.