An Embedded Hardware-Efficient Architecture for Real-Time Cascade Support Vector Machine Classification

Christos Kyrkou, Theocharis Theocharides
KIOS Research Center, Department of Electrical and Computer Engineering
University of Cyprus
Nicosia, Cyprus
{kyrkou.christos, ttheocharides}@ucy.ac.cy

Christos-Savvas Bouganis
Electrical and Electronic Engineering Department
Imperial College London
London, United Kingdom
christos-savvas.bouganis@imperial.ac.uk

***Abstract*—Support Vector Machines (SVMs) are considered as a state-of-the-art classification algorithm yielding high accuracy rates. However, SVMs often require processing a large number of support vectors, making the classification process computationally demanding, especially when considering embedded applications. Cascade SVMs have been proposed in an attempt to speed-up classification times, but improved performance comes at a cost of additional hardware resources. Consequently, in this paper we propose an optimized architecture for cascaded SVM processing, along with a hardware reduction method in order to reduce the overheads from the implementation of additional stages in the cascade, leading to significant resource and power savings for embedded applications. The architecture was implemented on a Virtex 5 FPGA platform and evaluated using face detection as the target application on 640×480 resolution images. Additionally, it was compared against implementations of the same cascade processing architecture but without using the reduction method, and a single parallel SVM classifier. The proposed architecture achieves an average performance of 70 frames-per-second, demonstrating a speed-up of 5× over the single parallel SVM classifier. Furthermore, the hardware reduction method results in the utilization of 43% less hardware resources and a 20% reduction in power, with only 0.7% reduction in classification accuracy.**

Keywords— Field Programmable Gate Arrays; Support Vector Machines; Cascade Classifier; Real-time and Embedded Systems; Parallel Architecture

I. INTRODUCTION

Support vector machines (SVMs) [1] are a powerful supervised pattern recognition algorithm which has been used in object detection, amongst other applications, demonstrating high classification accuracies [2-3]. However, for large scale problems the good classification accuracy rates of SVMs come with the cost of longer classification times. The reason for this is that the run-time complexity of the algorithm is proportional to the number of support vectors; which are samples from the training set that specify the separating hyperplane that the SVM algorithm selects in order to construct its classification decision function. As such, embedded SVM-based classification systems with hundreds of support vectors, find it difficult to meet real-time processing demands, without sacrificing accuracy. In order to speed-up the SVM classification process, the cascaded classification scheme has been proposed [4-8]. Under this scheme the majority of data are rejected in the first few stages resulting in significant speedups over single SVM classification. This approach has shown significant speedups over single SVM classification [6], and is highly suitable for embedded applications such as object detection where a single image generates a lot of data that needs to be classified.

However, achieving high performance without taking advantage of the inherit parallelism of SVMs is still challenging even with cascaded classification schemes. This has motivated a lot of research towards accelerating cascaded SVMs using parallel computing platforms such as Graphics Processing Units (GPUs) [9], and Field Programmable Gate Arrays (FPGAs) [10-15]. Implementations of SVMs on GPU platforms have shown great promise, however, GPUs have high energy consumption demands [13,20] and thus it is difficult to deploy them in embedded environments. Custom hardware architectures of SVMs, targeting embedded applications, have been implemented mostly on FPGAs [10-14], and are tailored to specific problems, while only considering single SVM classifiers. Hence, such hardware architectures are not optimized for processing cascaded SVM classifiers, as they do not take advantage of the throughput and processing demands of the different SVM stages in the cascade with respect to the available hardware resources.

This paper proposes a method for reducing the hardware implementation requirements of cascaded SVMs and a dedicated hardware architecture for cascaded SVM processing. The method involves rounding off the SVM training data (*support vectors* and *alpha coefficients*) of the early cascade stages to the nearest power of two values in order to replace multiplication operations with shifts with limited reduction in accuracy. The architecture features a hybrid processing scheme of sequential and parallel modules that exploit the cascade structure, in which classifiers at the beginning are used more frequently and

This work was funded by the following European Research Council and Cyprus Research Promotion Foundation projects: Project New Infrastructure Project/Strategic/0308/26, Project FP7/2007-2013/ERC grant agreement n^{o} 291508, and Project "EVAGORAS" of the INTERREG funding program.

have less computational demands than subsequent classifiers. The hardware architecture was implemented on a Virtex-5 FPGA platform, and evaluated using face detection on 640×480 resolution images. The implementation of the proposed hardware architecture for processing the adapted SVM cascade demonstrates an average performance of 70 frames-per-second, which is a 5× speedup over a parallel single SVM classifier implementation. Furthermore, by applying the hardware reduction method, the adapted hardware architecture for the implementation of the cascade consumes 43% less custom logic resources, and requires 20% less power, with only a 0.7% reduction in classification accuracy.

The paper is organized as follows. Section II provides the background on SVMs, classifier cascades, and related work. Section III details the hardware reduction method as well as the hardware architecture for cascade SVM processing. Section IV presents FPGA-based experimental results for performance, accuracy, area, and power. Finally, Section V concludes the paper.

II. BACKGROUND

A. Support Vector Machine Classification

A Support Vector Machine (SVM) is a supervised binary classification algorithm which tries to separate the data 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. The class samples that are on the boundary are called *support vectors* and influence the formation of the hyperplane [1]. The support vectors (SVs) are obtained during the SVM training phase, and correspond to non-zero *alpha coefficients* derived from the training optimization problem [1]. They are used to form the classification model by which SVMs classify new input data. SVMs also utilize a technique called the *kernel trick* [1], which is a method of projecting data into a higher dimensional space though a *kernel function*, without the need to explicitly use a mapping function. This projection allows for linear separation of data in a higher dimensional space. The classification decision function for SVMs is given in (1), in which a_i are the alpha coefficients, y_i are the class labels of the support vectors, s_i are the support vectors, z is the input vector, $K(z,s_i)$ is the chosen kernel function, and b is the bias. The support vectors correspond to training set samples which have non-zero alpha coefficients.

The computational demands of SVM classifiers depend on the choice of kernel. For linear SVMs (2) the kernel is replaced by a dot-product operation between the input data and a feature vector which is computed directly from the support vectors. However, in the case of non-linear SVMs (3)-(4), the kernel is a more complex function and the feature vector cannot be directly obtained. Hence, the input data needs to be processed with all support vectors before a classification outcome can be obtained. To reduce the computational demands of non-linear kernels different techniques have been proposed. One such method is the reduced-set-method by Burges [17], which tries to find a smaller set of vectors, called *reduced-set-vectors,* in order to approximate the decision function of the full SVM retaining most of the classification capabilities [8]; resulting in a reduced-set-vector-machine (RSVM).

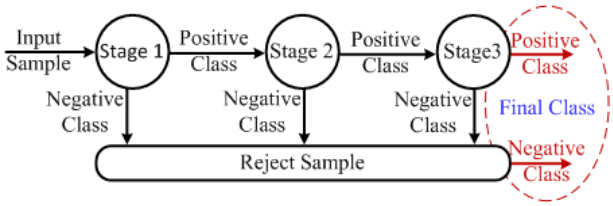

Fig. 1. Cascade Classifier Scheme

$$D(z) = sign(\sum_{i=1}^{N_{sv}} a_i y_i K(z, s_i) - b) \quad (1)$$

Linear: $K(s,z) = s \bullet z$ (2)

Polynomial: $K(s,z) = ((s \bullet z) + const)^d, d > 0$ (3)

RBF: $K(s,z) = \exp(\|s - z\|^2 / (2\sigma^2))$ (4)

B. Cascaded Support Vector Machine Classification

It is possible to speed-up SVM-based classification systems for a variety of applications, including object detection, by exploiting the facts that: (a) the majority of the samples presented to the classifier do not belong to the object class and (b) the majority of those samples can be easily distinguished from samples belonging to the object class. Cascade SVM classifiers [6-10], take advantage of these two observations by utilizing stages of classifiers, which are sequentially applied to the input data. The early stages, usually linear SVMs (3), have low complexity, meaning that they require less training data to be processed, and as such take less time to process. Conversely, the latter stages, typically kernels (4)-(5), have higher complexity as they require more training data to be processed, and hence have a longer classification-time. If at any stage the input data is classified as a non-object the classification process stops and the next sample is processed, otherwise it must go all the stages to be classified as an object (Fig. 1). Under this scheme the stages at the beginning of the cascade discard a large amount of input samples very fast, resulting in significant speedups. In addition, it is also possible to use the reduced-set-method [17], to reduce the number of support vectors required by the non-linear kernel stages in order to further improve classification times.

C. Related Work

Hardware architectures for SVM classification have been proposed in order to take advantage of the inherit parallelism of the SVM computation flow. The majority of proposed hardware architectures attempt to improve performance by employing parallel processing modules which process the elements of the input vector in parallel [10-11]. Such architectures dependent on the vector dimensionality in terms of computational resources, and thus, suffer from scalability issues when dealing with applications with high dimensional vectors. More generic

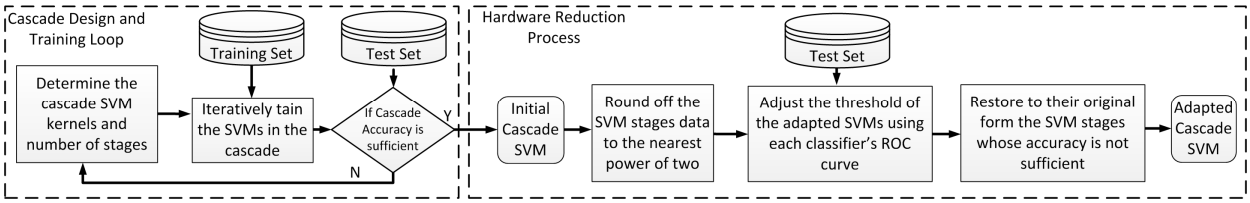

Fig. 2. Cascade Hardware Reduction Method: The initial cascade SVM obtained after training is adapted to reduce its hardware processing requirements

architectures have also been proposed, that utilize arrays of processing elements to parallelize the vector processing [12, 13]. Such, architectures attempt to provide a generic platform for SVM processing, and use FPGAs as coprocessors to CPU-based systems. However, the proposed architectures were developed for specific problems and thus are not easily extendable to other scenarios, such as cascaded classification. In addition, there has also been research has also focused on simplifications to make SVM classification more hardware friendly and suitable for devices with limited computational resources. These approaches include using CORDIC algorithms to compute the kernel functions [14], however, the iterative operations of these algorithms makes it challenging to achieve high performance for applications that require high data throughput such as object detection. Furthermore, other works [16] proposed that the computations be done in the logarithmic number system so that all multiplications are substituted by additions. It is unclear, however, what overheads are introduced by the number system conversions for a fully parallel implementation, since [16] presented a sequential implementation. The work in [15] proposes an optimized architecture which utilizes processing modules with different bitwidth requirements for each stage of the stage in the cascade SVM classifier. This results hardware savings for the implementation of the cascade, however, it is primarily directed towards classification problems where different vector elements have different bitwidth requirements in order to take advantage of custom arithmetic optimizations.

NVidia's compute unified device architecture (CUDA) has been used in [9] in order to speedup SVM classification using the parallel computing resources of a GPU showing improved results compared to CPU implementations. However, GPUs are power hungry devices compared to FPGAs [20], (FPGAs consume approximately an order of magnitude less power as shown in [13]) and as such they are not suitable for embedded applications.

From the works found in the literature only [15] has looked into cascade SVM implementation focusing, however, on heterogeneous classification problems. Furthermore, the benchmarks used in each work are different, and as such it is difficult to compare other works. Considering, the above this paper proposes a hardware architecture for cascaded SVM classification, and optimized I/O flow, along with a generic method to reduce the hardware requirements for the implementation of the cascade. The proposed framework is generic and flexible and can thus be used by other works as well in order to improve certain aspects of hardware architectures for SVM processing.

III. Proposed Hardware Architecture and Hardware Reduction Method

A. Cascade SVM Hardware Reduction

A cascade SVM is made up of SVMs with different computational complexity. The stages at the beginning of the cascade construct simple decision functions and as such require processing less support vectors than subsequent stages. This is because the objective of the low complexity SVMs is to guarantee that the positive samples will go through to the final stage while a large amount of negative samples will be discarded rather quickly. In contrast, subsequent stages need to be more accurate and as such construct more complex decision functions which have increased computational complexity. As such, it is possible to adapt the low complexity SVM stages, in order to make them more hardware friendly, while still able to discard a a large amount of negative samples.

The proposed hardware reduction method is to round-off the support vector and alpha values of the low complexity kernels with the nearest power of two values. This will result in all the multiplication operations in the SVM classification phase (the kernel dot-product calculations and computations related to the alpha coefficients) becoming shift operations. Additionally, since the support vectors 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). Hence, this results in an *adapted cascade SVM* with reduced storage and computational demands. However, by approximating the support vectors and alpha coefficients the resulting classification accuracy will be different from that of the *initial SVM cascade*. The receiver-operating-characteristic (ROC) curve of each cascade stage 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 and false positive rates, given a test set. As such, by setting the appropriate threshold the performance of the adapted SVM cascade stages can be adjusted to match the true positive rate of the initial SVM cascade stages. However, there are tradeoffs which stem from changing the original classification model. The tradeoff for the reduced computational and storage demands is an increase in the false positive rate of the adapted classifiers. However, given that a large number of

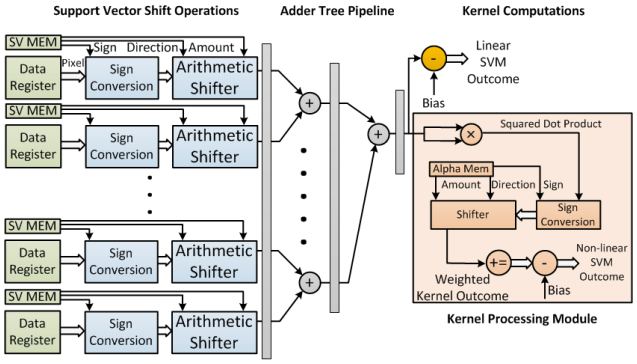

Fig. 3. Parallel Processing Module Handles the processing of the nearest power of 2 adapted SVM stages. The shift units and adder tree are used by all kernels while only non-linear kernels use the kernel module.

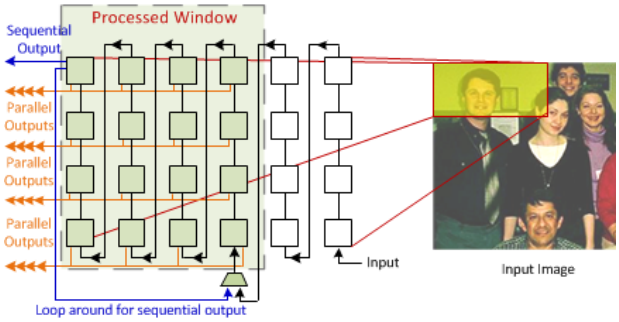

Fig. 4. Shift Register Structure used to provide efficient I/O with two modes of operation (parallel and sequential) to accommodate the needs of the cascade stages.

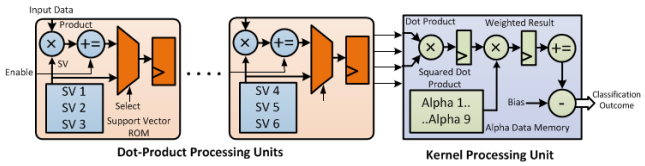

Fig. 5. The Sequential Processing Module Architecture: Consists of two processing units: The dot-product processing units handle the dot-product computation, and the kernel processing unit, which is shared amongst the dot-product units, handles the kernel-related operations.

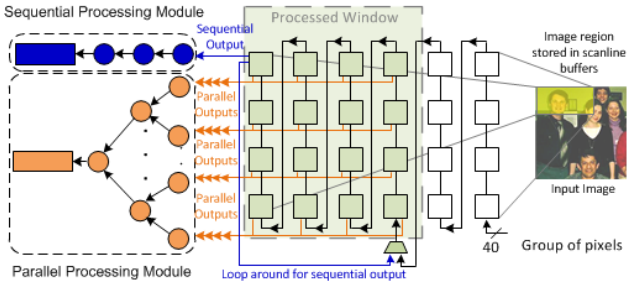

Fig. 6. System Architecture

samples will still be rejected, the increase is not that significant. Finally, the higher complexity SVMs, typically only the final stage, are not approximated and are used in their initial form as any approximations would have a significant impact on their classification accuracy. The process is summarized in Fig. 2.

The hardware reduction process is done after the cascade structure is decided, meaning that the kernel function, and amount of support vectors or reduced-set-vectors 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 dependent on the specific hardware architecture used for the implementation of the cascade and as such can be optimized to fit different architecture requirements.

B. Cascade Support Vector Machine Architecture

The nature of the cascaded SVM classifiers means that each stage will have less input data to process and more support vector data to process than the previous. Hence, efficient hardware architectures need to take into consideration the throughput and processing needs of each stage in the cascade. Accordingly then, the proposed hardware architecture (Fig. 6) for the cascaded SVM classifier consists of two main processing modules, which provide different parallelism with respect to the input data, in order to meet the different demands of the cascade stages. The first is a fully parallel module which performs the processing necessary for all the adapted SVM stages (Fig. 3) and the parallelism focuses on processing all the input vector elements in parallel. The second is a sequential processing module (Fig. 5), optimized for the high complexity SVM stages which demand processing a large number of support vectors and thus parallelism focuses on processing more support vectors in parallel. In addition, a shift register structure (Fig. 4) is used to provide sequential and parallel data access to the two processing modules, and also to take advantage of potential data overlap and reduce memory I/O.

1) Parallel SVM Processing Module

The parallel SVM processing module handles the processing of the low complexity SVM stages which have been adapted using the proposed hardware reduction method. Specifically, the proposed architecture can process linear and 2nd degree polynomial kernels, but can be used for other kernels with minor modifications. The characteristic of the low complexity stages is that they require processing only a few support vectors. As such, to achieve high performance, it is preferable to process input vector elements in parallel. The architecture is comprised of parallel sign conversion and shift units, and a tree of adders, in addition to a kernel processing unit, which is also, implemented using shifters. The shift data are fetched in parallel from small ROMs, and include the sign of the support vector used to convert the input pixel to a positive or negative two's complement format, the shift amount, and finally the direction of the arithmetic shift operation.

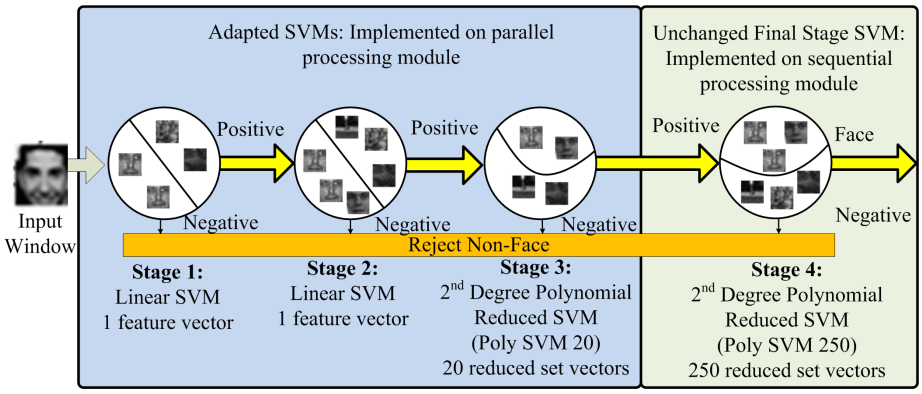

Fig. 7. Cascaded SVM architecture with four stages. The first three stages have been rounded-off to the neatest power of 2 values, while the final stage remained unchanged. Each stage classifies the input as belonging to the positive or negative class. In the case that it is positive it goes on to the next stage for verification, else it is discarded.

The operation of the parallel processing module 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 ROMs. The partial results are added together using a pipelined tree of adders so that the dot-product outcome can be obtained. In the case of linear kernels, subtracting a bias value from the dot-product outcome will suffice in order to obtain the classification result. However, for 2nd degree polynomial kernels, the kernel computation module handles the latter steps of the classification phase. Only one multiplier is used in the parallel processing module and is used to perform the square operation. The processing of the alpha coefficients is done with a sign conversion unit and a shift unit similarly to the processing of the support vectors. An accumulator is used to accumulate the result, and a subtractor to process the bias. The parallel processing module is pipelined, so one support vector enters the pipeline every cycle.

2) Sequential SVM Processing Module

The sequential SVM processing module is responsible for performing the processing necessary for the final SVM stage. This final stage will most likely process only a small percentage of the input data; however, it will have the most support vectors. As such, instead of processing the input vector in parallel the focus is on processing more support vectors in parallel. This is achieved with the architecture shown in Fig. 5, and is comprised of a series of pipelined processing elements. The majority of the units in the module are vector processing units and each unit handles the dot-product for one support vector with the input vector. They are comprised of a multiply-accumulate unit, and also a ROM which contains the data for one or more support vectors. The final unit in the pipeline is the kernel processing unit which is equipped with multipliers and accumulators to carry out the scalar processing of the SVM processing flow. The input vector is processed with a group of support vectors at a time, and each vector processing unit handles the processing of one support vector. Once a group of support vectors is processed the next group follows. Hence, the size of the pipeline can be adjusted to fit the available hardware budget by adjusting the number of support vector groups. Each vector processing unit in the pipeline processes one support vector with the input vector at a time. The input vector pixels are propagated from one unit to the next so processing happens in parallel. When the processing is done, the results are transferred sequentially through the pipeline and are processed by the kernel processing module which evaluates the kernel for each support vector, and also processes it with the alpha coefficients. Once all support vectors are processed, the final classification result is obtained by subtracting the bias from the accumulated result.

3) Cascade Optimized I/O and Processing Flow

The different throughput requirements of the cascade SVM processing modules require an I/O mechanism that can adjust to the different needs of each module; that is parallel as well as sequential data transfer. Furthermore, it should take advantage of the application-specific characteristics to facilitate data reuse and reduce memory accesses. To illustrate the above features we consider the design of such a structure for object detection applications. An optimized I/O mechanism for object detection can be developed based on an array of shift registers that incorporates the above features and also acts as local storage for the image segment that is currently being processed (Fig. 4). The input image pixels enter the scanline buffers and are propagated column-wise into the structure. The image region that is at the left-most part of the scanline buffer is the window that is currently being processed by either the sequential or the parallel processing modules. As such, 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 leftmost region of the scanline buffer and is ready to be processed next. The data flow of the left-most registers changes depending on whether the data are fed to the parallel or the sequential processing module. In the case of the parallel module the window data are outputted in parallel. On the other case, the registers form a chain of shift registers so that data are outputted sequentially for the sequential processing module, from the leftmost top row register. Furthermore, during sequential

output operation, the window data are looped back to the scanline buffers, using a multiplexer on the start of the chain, so that the window is formed again. This is required in the case where the vector processing units of the sequential processing module are less than the support vectors, and thus the same window must be processed with a new group of support vectors.

IV. Experimental Platform and Results

The proposed hardware architecture is evaluated using the embedded application of face detection on 640×480 resolution images. It is evaluated in terms of the detection accuracy, frame-rate, power consumption, as well as requirements in terms of computing resources. A cascade of four SVM stages, illustrated in Fig. 7, was trained in MATLAB and was used for evaluation of the architecture and reduction method. Additionally, the proposed architecture, which will be referred to as the *adapted cascade,* is compared against two other systems in terms of the above metrics. The first system, which will be referred to as the *initial cascade* is an implementation of the architecture that can process the same cascade SVM, but without applying the hardware reduction method, and thus the parallel processing module is implemented using multipliers. The second system, referred to as *single parallel classifier*, implements a parallel version of the final cascade stage (PolySVM250, Fig. 7, SVM stage 4), where the

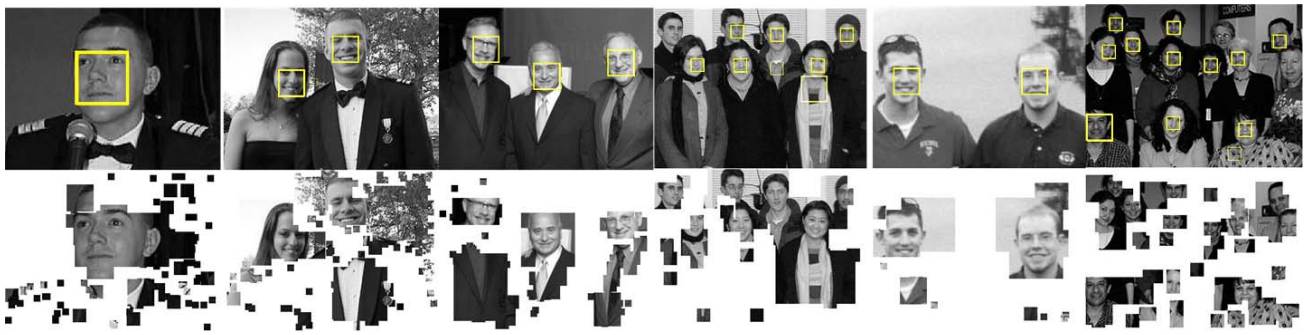

Fig. 8. Top: Detection Results on 640×480 images. Bottom: Images made up of the windows (~100) that reach the final cascade stage

TABLE I. Detection Accuracy of Cascade Implementations

Cascade Stages	Initial Cascade SVM		Adapted Cascade SVM	
	Detection Accuracy		Detection Accuracy	
	True Positive	False Positive	True Positive	False Positive
1. Linear SVM	94.6 %	21.1 %	96 %	33.2 %
2. Linear SVM	95.8 %	19.7 %	98 %	35 %
3. Poly SVM 20	95.8 %	12.5 %	95 %	16.8 %
4. Poly SVM 250	87.2 %	0.3 %	86.9 %	0.2 %
SVM Cascade	84.7 %	0.2 %	84 %	0.2 %

TABLE II. Window Statistics for each Stage in the Cascade

Cascade Stages	Average Number of windows processed	Rejection Rates	% of remaining windows after each classifier
1. Linear SVM	14 712	95%	-
2. Linear SVM	779	22%	5%
3. Poly SVM 20	611	68%	4%
4. Poly SVM 250	196	-	1%

TABLE III. FPGA Resource Utilization for all three Implementations

FPGA Resources	Single Classifier	Adapted Cascade SVM	Initial Cascade SVM	I/O System
Slice LUTs (69 120)	61 519 (89%)	31 854 (46%)	62 055 (89%)	7 016 (10%)
Slice Registers (69 120)	21 171 (30%)	13 038 (18%)	15 474 (22%)	8 180 (11%)
DSP48E (64)	54 (84%)	59 (92 %)	59 (92%)	3 (4%)
Block Ram (148)	131 (88%)			15 (10%)
Frequency	84 MHz			

TABLE IV. Memory Requirements of Cascade Implementations

Cascade Stages	Initial Cascade SVM			Adapted Cascade SVM		
	Bit Representation			Bit Representation		
	SVs	Alpha	Bytes	SVs	Alpha	Bytes
1. Linear SVM	10	-	500	6	-	300
2. Linear SVM	10	-	500	6	-	300
3. PolySVM 20	10	10	50	6	6	30
Total Memory			1050,5			630

sequential processing module (shown in Fig. 5) is replicated as many times as possible on the FPGA. Thus up to six sequential parallel modules are instantiated in order to increase parallelism, so that it is possible to process six successive windows from the image in parallel. All three systems were evaluated using a Xilinx ML505 board equipped with a Virtex 5-LX110T FPGA and compared in terms of performance, detection accuracy, custom logic resource utilization, and power consumption for the application of face detection on 640×480 images. A Microblaze-based system was used for I/O and verification purposes, while for all three systems a memory capable of storing a 640×480 image and a scanline buffer where used to store the input image and process. The following sections detail the evaluation process and the results.

A. Detection Accuracy

The SVM cascade (shown in Fig. 7) was comprised of two linear kernel SVMs and by two 2nd degree polynomial kernel SVMs and is capable of processing windows of 20×20 pixels, resulting in 400-dimensional vectors. The training process for the cascade was similar to the one in [5]. The two polynomial SVMs where reduced to 20 reduce-set-vectors for the third stage (PolySVM20), and 250 reduce-set-vectors for the final stage (PolySVM250). The accuracy of the adapted cascade SVM was tested on a widely used test dataset from [18] consisting of 2,429 positive and 23,573 negative 20×20 samples and compared to that of the initial cascade SVM. As shown in Table I the adapted SVMs after a hardware reduction method have similar accuracy to that of the initial SVMs in terms of true positive detection accuracy. This means that the majority of object samples that would go through in the initial cascade will go through in the adapted cascaded as well. However, the false positive rate has been negatively impacted. This is to be expected since the approximations introduced a discrepancy between the initial and adapted SVM models. Overall, however, the final detection accuracy of the adapted cascade is very close to that of the initial cascade. This is evident from Table II which shows the average number of windows that reach each stage for a set of 135 images from [19] resized to 640×480 resolution (detection results of which are shown in Fig. 8). Each 640×480 image generates a total of 14712 windows, for all scales, all of which are processed by the first SVM stage. All generated windows are processed by the first SVM stage; however, around 1% of those reach the final stage. Overall, the detection accuracy is of similar to other works which range from 75% to 88% [4-8].

B. Hardware Resource Utilization

The two cascade implementations (initial and adapted) have the same architecture and data flow. Specifically, the parallel module processes all the elements of a 400-dimensional input vector (corresponding to a 20x20 window) in parallel; on the other hand the sequential processing module processes groups of 50 support vectors at a time. As such, it takes 5 iterations to process all the support vectors of the final SVM stage. The only difference between the two implementations is that in the adapted cascade case the parallel processing module was optimized using the hardware reduction method. Consequently, the multiplication units where replaced with shift units and the data stored in the training data ROMs correspond to shift values instead of support vector values. Through this optimization the total memory demands to store the SVM training data for the first three stages of the SVM cascade are reduced by 40% (Table IV). Considering both the reduction in processing and storage resources together the adapted cascade implementation requires 43% less FPGA Look-Up Table (LUT) resources compared to the initial cascade implementation, as shown in Table III. All three implementations utilize the same number of DSPs for different purposes. In the two cascade implementations there are utilized only by the sequential processing module while for the single parallel implementation they are used only for the kernel modules.

TABLE V. SVM CASCADE PROCESSING CYCLES PER STAGE

Cascade Stages	Processing cycles needed to process a single input	Processing Unit
Stage 1: Linear SVM	11	Parallel Processing Module
Stage 2: Linear SVM	11	
Stage 3: Poly SVM 20	31	
Stage 4: Poly SVM 250	2535	Sequential Processing Module

C. Frame Rate

In order to measure the frame rate of the three implementations a set of 135 640×480 images were used. to find the average frame-rate. Each frame requires a different time to be processed, for the cascaded implementations, depending on how many windows reach each stage, and by how many cycles it takes a stage to process an input (shown in Table V for the cascade under consideration). In contrast, the single parallel SVM classifier takes the same time to processes all frames. In addition to the actual processing time, the I/O delays per frame also negatively impact performance. In order to achieve higher performance, I/O and memory operations such as filling the scanline buffers, overlap with window processing. Overall, both the adapted cascade and initial cascade implementations achieve a performance of 70 fps, resulting in a 5× speedup over the single parallel SVM classifier implementation which achieved 14 fps, despite processing six windows in parallel. The high performance of the cascade SVM implementations is attributed primarily to the fact that most windows are discarded by the first two cascade stages which and take only a few cycles to classify a window.

D. Power Consumption

We used power analysis tools from Xilinx in order to perform an initial analysis on the power consumption demands of the three SVM implementations. The single parallel classifier requires 4.2W, the adapted SVM cascade 3.2W, and the original SVM cascade 4.1W. Power consumption drops down to 3.2W from 4.1W resulting in a

20% reduction in power consumption stemming from the reduction in hardware resources. It must be noted, that in these initial results it is assumed that the parallel and sequential processing modules in the cascade have the same utilization rate. However, the sequential processing module is only used for a small amount of input data (as shown in Table II). It is anticipated that with a more detailed analysis the reduction will be much greater. Nevertheless, even with this rather pessimistic scenario the reduction is still significant.

E. Discussion and Impact

The proposed hardware reduction method and processing architecture combine into an efficient framework for the design of cascade SVM classifiers for embedded applications. Specifically, the framework has resulted in improvements over both a typical cascade SVM implementation as well as a single SVM with parallel processing capabilities. The framework is generic enough so that other SVM implementations can use it to improve on different aspects depending on the requirements of the targeted application. Works in the literature that proposed hardware architectures for the implementation of SVMs include [10], [11], [12], [13], and [14]. Such hardware implementations can utilize a linear SVM, that is adapted with the method shown in this paper, to act as a pre-processor to their proposed architectures. In this way the performance can be improved without a major impact on hardware implementation requirements. Moving forward the proposed framework can be further improved by incorporating ideas proposed in [15] for exploiting custom precision bitwidth for heterogeneous training sets. This is especially useful for the implementation of fully unrolled SVMs where each vector element can be optimized independently. Furthermore, the targeted architectures for parallel and sequential processing can be tweaked in terms of available parallelism in order to meet application-specific constraints, and thus it is also possible to use similar architectures proposed in other works for the implementation of each stage in the cascade.

V. CONLCLUSIONS

This paper presented a hardware architecture for SVM cascade processing, as well as a hardware reduction method based on adapting the training data of the early stages in the cascade, to allow for a more efficient implementation of cascaded SVMs for embedded applications. The approaches were verified experimentally on a Virtex 5 FPGA using the application of face detection. Specifically, the cascade architecture was capable of processing 70 640×480 frames per second, demonstrating a 5× speedup over a single parallel SVM classifier implementation. In addition, the proposed hardware reduction method results in a cascade implementation which requires 43% less custom logic resources and 20% less power, with minimal reduction in accuracy. The results are encouraging and show the potential of the proposed framework to be used for the implementation of efficient hardware-based cascade SVM classifiers for embedded applications.

REFERENCES

[1] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, no. 3, 1995, pp. 273-297.

[2] E. Osuna, R. Freund, and F. Girosi, “Training support vector machines: an application to face detection,” IEEE Conference on Computer Vision and Pattern Recognition, 1997, pp. 130-136

[3] B. Heisele, T. Poggio, M. Pontil, "Face detection in stillgray images," A.I. Memo 1687, Center for Biological and Computational Learning, MIT, Cambridge, MA, 2000.

[4] S. Romdhani, P. Torr, and B. Schlkopf. "Efficient face detection by a cascaded support-vector machine expansion," Royal Society of London Proceedings Series A, 460:3283–3297, November 2004.

[5] B. Heisele, T. Serre, S. Prentice, and T. Poggio. "Hierarchical classification and feature reduction for fast face detection with support vector machines,: Pattern Recognition, 36:2007–2017(11), Sept. 2003.

[6] Ma, Y., X. Ding, "Face Detection Based on Cost-sensitive Support Vector Machines," Proceedings of the First International Workshop on Pattern Recognition with Support Vector Machines, Lecture Notes in Computer Science, Vol. 2388, London, Springer, 260–267, 2002.

[7] Sahbi, H., Boujemaa, N., "Coarse-to-fine support vector classifiers for face detection," Pattern Recognition, 2002. Proceedings. 16th International Conference on , vol.3, no., pp. 359- 362 vol.3, 2002.

[8] Kukenys, I.; McCane, B.; , "Classifier cascades for support vector machines," Image and Vision Computing New Zealand, 2008. IVCNZ 2008. 23rd International Conference, pp.1-6, 26-28 Nov. 2008.

[9] B. Catanzaro, N. Sundaram, and K. Keutzer, "Fast support vector machine training and classification on graphics processors," 25th international conference on Machine learning, 2008, pp. 104–111.

[10] I. Biasi, A. Boni, and A. Zorat, "A reconfigurable parallel architecture for SVM classification," 2005 IEEE International Joint Conference on Neural Networks, 2005. IJCNN'05. Proceedings, 2005, pp. 2867-2872.

[11] O. Pina-Ramirez, R. Valdes-Cristerna, and O. Yanez-Suarez, "An FPGA implementation of linear kernel support vector machines," IEEE International Conference on Reconfigurable Computing and FPGA's, 2006, pp. 1–6.

[12] R. Roberto, H. Dominique, D. Daniela, C. Florent, and O. Salim, "Object Recognition System-on-Chip Using the Support Vector Machines," EURASIP Journal on Advances in Signal Processing, vol. 2005, 1900, p. 993–1004.

[13] P.H. Graf, Srihari, C. Durdanovic, V. Jakkula, M. Sankardadass, E. Cosatto, and S. Chakradhar, "A Massively Parallel Digital Learning Processor," *NIPS*, 2008, pp. 529-536.

[14] M. Ruiz-Llata, G. Guarnizo, M. Yébenes-Calvino, "FPGA implementation of a support vector machine for classification and regression," The 2010 International Joint Conference on Neural Networks, vol., no., pp.1-5, 18-23 July 2010.

[15] Papadonikolakis, M.; Bouganis, C.; , "Novel Cascade FPGA Accelerator for Support Vector Machines Classification," IEEE Trans. on Neural Networks and Learning Systems, vol.23, no.7, pp.1040-1052, July 2012.

[16] F. Khan, M. Arnold, and W. Pottenger, "Hardware-based support vector machine classification in logarithmic number systems," IEEE International Symposium on circuits and systems, May 2005, p. 5154.

[17] C. J. C. Burges. "Simplified support vector decision rules," In International Conference on Machine Learning, pages 71–77, 1996.

[18] “CBCL Face Database #1” [Online]. Available: http://cbcl.mit.edu/software-datasets/FaceData2.html

[19] Bao Face Database, [Online]. Available: http://www.facedetection.com/facedetection/datasets.htm

[20] Jeremy Fowers, Greg Brown, Patrick Cooke, and Greg Stitt. 2012. A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications. FPGA '12. ACM, New York, NY, USA, 47-56.