Int J Parallel Prog (2018) 46:1220–1246
https://doi.org/10.1007/s10766-017-0514-1

Boosting the Hardware-Efficiency of Cascade Support Vector Machines for Embedded Classification Applications

Christos Kyrkou[1] · **Theocharis Theocharides**[1] · **Christos-Savvas Bouganis**[2] · **Marios Polycarpou**[1]

Received: 30 April 2015 / Accepted: 16 June 2017 / Published online: 23 June 2017

Abstract Support Vector Machines (SVMs) are considered as a state-of-the-art classification algorithm capable of high accuracy rates for a different range of applications. When arranged in a cascade structure, SVMs can efficiently handle problems where the majority of data belongs to one of the two classes, such as image object classification, and hence can provide speedups over monolithic (single) SVM classifiers. However, the SVM classification process is still computationally demanding due to the number of support vectors. Consequently, in this paper we propose a hardware architecture optimized for cascaded SVM processing to boost performance and hardware efficiency, 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. The architecture was evaluated for the application of object detection on 800 × 600 resolution images on a Spartan 6 Industrial Video Processing FPGA platform achieving over 30 frames-per-second. Moreover, by utilizing the proposed hardware reduction method we were able to reduce the utilization of

✉ Christos Kyrkou
kyrkou.christos@ucy.ac.cy

Theocharis Theocharides
ttheocharides@ucy.ac.cy

Christos-Savvas Bouganis
christos-savvas.bouganis@imperial.ac.uk

Marios Polycarpou
mpolycar@ucy.ac.cy

1 Department of Electrical and Computer Engineering, KIOS Research Center, University of Cyprus, Nicosia, Cyprus

2 Electrical and Electronic Engineering Department, Imperial College London, London, UK

FPGA custom-logic resources by ∼30%, and simultaneously observed ∼20% peak power reduction compared to a baseline implementation.

Keywords Field Programmable Gate Arrays (FPGAs) · Support Vector Machines (SVMs) · Cascade classifier · Real-time and embedded systems · Hardware architecture · Parallel processing

1 Introduction

Support Vector Machines (SVMs) have been widely adopted since their introduction by Cortes and Vapnik [1], and are now considered one of the most powerful classification engines due to their mathematical background that is based on statistical learning and is able to accurately model complex classification boundaries. Consequently, there has been growing interest in utilizing SVMs in numerous applications, including visual object detection systems, demonstrating high classification accuracies [2–4]. However, for large-scale problems the good classification accuracy rates of SVMs come with the cost of longer classification times, as the classification complexity of SVMs is proportional to the number of samples needed to specify the separating hyperplane between classes, referred to as support vectors (SVs). As such, SVM-based classification systems with hundreds of support vectors, find it difficult to meet real-time processing demands, without sacrificing accuracy, especially for embedded applications such as object detection with thousands of data that need to be classified [4]. Relevant literature on SVMs [5–7] suggests a cascaded classification structure in order to speed-up the SVM classification process for a class of the aforementioned applications where the majority of data that need to be classified belongs to one of the two classes. In the cascade classification approach, multiple SVM classifiers are arranged in stages of increasing computational complexity and accuracy. The early stage classifiers are computationally less demanding and are responsible for the removal of a large amount of negative class data, which do not exhibit similar patterns to the positive samples. On the other hand, the latter stages have higher accuracy and thus higher computational complexity, in order to be able to distinguish between similar samples belonging to different classes. However, they only classify the samples that successfully pass the previous stages. Consequently, when utilizing the cascade approach significant speedups over monolithic (single) SVM classification are possible [5,7]. Still, when considering embedded applications (e.g. automotive, home entertainment, and robotics) with real-time and power consumption constraints and limited resources, the design of SVM-based classification systems that process hundreds of support vectors and need to classify a large number of instances, is still challenging to achieve.

Recent advances in hardware acceleration of SVMs feature extensive use of parallel computing platforms such as Graphics Processing Units (GPUs), and reconfigurable hardware fabric (Field Programmable Gate Arrays—FPGAs in particular) [8–10]. Relevant research has produced quite promising results in terms of performance for use in embedded systems. However, even if implementations of SVMs on GPU platforms have gained considerable attention due to the high-level programming capabilities,

GPUs still face challenges with regards to power consumption, especially with the increasing development of portable resource-limited platforms, requiring specific hardware solutions and large scale problems [11,12]. Hence at present, computing systems based on FPGAs and customized hardware accelerators allow to exploit the inherit parallelism of algorithms such as SVMs, whilst achieving efficient implementation suitable for real-time processing and low-power operation. Furthermore, FPGA platforms provide flexibility and hence allow hardware/software co-design techniques. Justifiably then, there has been a considerable amount of research work on SVM hardware architectures. However, existing hardware architectures proposed for the acceleration of SVMs, consider only monolithic classifiers, which are not optimized to efficiently handle problems, where the majority of data belong to one of the two classes. As such, designing specialized hardware accelerators for multistage cascade SVMs based on existing approaches is a challenging task, especially due to the increase in the number of classifiers and subsequent hardware complexity, and their different computational demands, which require flexibility, low power, real-time operation, and often with limited available resources.

This work extends and improves our preliminary research in [13] which presented a hybrid hardware architecture that exploited the cascade SVM flow, where classifiers at the beginning are used more frequently than subsequent stages, to provide a hardware-efficient implementation capable of real-time classification while outperforming a monolithic SVM classifier. In this work, we further elaborate on the advantages of the proposed architecture first presented in [13], and demonstrate its applicability for embedded applications by evaluating it on larger-scale benchmark applications with different data sets and computational demands. We also outline the trade-offs of the proposed design optimization method that is based on approximating the support vectors with power of two values in order to replace multiplications with shift units and reduce the resource requirements. In addition, a feature extraction mechanism based on the hardware-efficient local binary patterns (LBPs), is selectively incorporated into the architecture in order to improve the accuracy of the SVM cascade for object detection applications. The cascade architecture optimized with the proposed hardware reduction method is implemented as part of a complete classification system on a Spartan-6 industrial video processing FPGA platform. The system was evaluated on a larger test set and higher resolution images (800×600) than our previous work for the applications of face and pedestrian detection. As it will be shown in Sect. 4.6 the proposed system achieves over 30 frames-per-second (fps), which is capable for real-time video processing, while processing more windows than other works, with 80% detection accuracy which is on par with cascade SVM software implementations for the targeted applications [5–7]. Furthermore, the hardware reduction method resulted in the utilization of $\sim$30% less FPGA logic resources and reduction of peak power by $\sim$20%.

The paper is organized as follows. Section 2 provides the background on SVMs, cascade classifiers, object detection, and related work on the hardware acceleration of SVMs. Section 3 details the hardware architecture for cascade SVM processing and the hardware reduction method. Section 4 presents FPGA-based experimental results on the achieved frame-rate, detection accuracy, and resource utilization trade-offs, for face and pedestrian detection, as well as comparison with related works. Finally, Sect. 5 concludes the paper.

2 Background

2.1 Support Vector Machines (SVMs)

A Support Vector Machine (SVM) is a supervised binary classification algorithm which maps data into a high-dimensional space where an optimal separating hyperplane is constructed [1,2]. 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 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 tries to separate the data samples of two different classes, by finding the hyperplane with the maximum margin from the training samples that lie at the boundary of each class (Fig. 1a). The training samples that are on the boundary are called *support vectors (SVs)* and influence the formation of the hyperplane [1,2]. The support vectors obtained during the SVM training phase, correspond to non-zero *alpha coefficients* derived from the training optimization problem [2], and constitute the SVM classification model with which to classify new input data. 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. 1b), 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 (CDF) for SVMs is given in (1), where N_s is the number of support vectors obtained from training, a_i are the alpha coefficients, $\mathrm{y_i y_i}$ are the class labels of each sample, 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.

$$\mathrm{C(z)}: \ \mathrm{sign}\left(\sum_{\mathrm{i}=1}^{\mathrm{N_S}} \alpha_\mathrm{i} \mathrm{y_i K\,(z,\, s_i)} + \mathrm{b}\right) \tag{1}$$

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

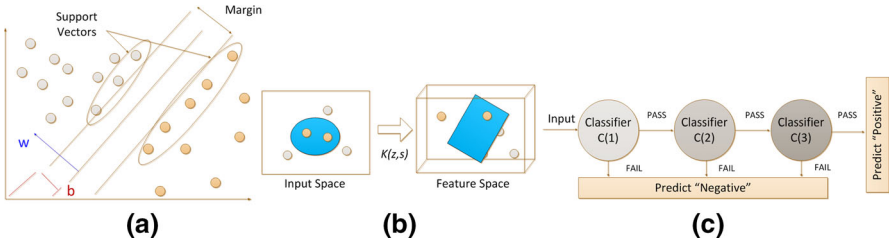

Fig. 1 **a** SVM concepts: *separating hyperplane*, *support vectors*, *normal vector w*, *bias*, and *margin*. **b** The kernel Trick visualization. **c** Cascade classification scheme overview

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

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

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

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. 1a), and is computed directly from the support vectors using $w = \sum_{i=1}^{N_s} y_i a_i s_i$. 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 from the support vectors. Hence, the input vector needs to be processed with all support vectors, and the kernel-specific operations need to be performed, before a classification outcome can be obtained. To reduce the computational demands of non-linear kernels the reduced-set-method [14] has been proposed, 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 [7]; which yields a *reduced-set-vector-machine* (RSVM). This results in losing the association with the original training problem, i.e. for image classification problems the SVs are not images any more.

2.2 Cascade Support Vector Machines

In many applications, achieving linear separation of a data set is quite rare. Hence, non-linear kernels are necessary in order to obtain accurate classification results; however, classification speeds can be slow with such kernels. It is possible to accelerate SVM-based classification for a certain class of applications, such as video object detection, that exhibit the following characteristics: (a) the majority of the instances presented to the classifier belong to one of the two class and (b) the majority of those instances are not similar to the instances of the other class. Based on these characteristics software implementations in the literature [15,16] have tried to take advantage of these two observations by utilizing stages of SVMs of increasing complexity, which are sequentially applied to the input data (Fig. 1c). Such structures mostly follow a cascade structure [5–7] where SVMs of increasing complexity are arranged in a hierarchy of stages. The stages can be separate classifiers or the decomposition of one larger classifier. Regardless, the commonality of these structures is that the SVM stages at the beginning of the hierarchy [usually linear kernels (2)] have lower computational complexity (i.e. need to process only a small number of SVs) and are tasked with removing the majority of samples from the negative class. The latter stages [usually kernels (3), (4)] then are able to perform more accurate classification on the remaining samples, which, however, incurs a higher computational cost (i.e. need to process more SVs). Hence, an input sample needs to pass all stages to be classified as positive (Fig. 1c), otherwise it is classified as negative. 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. Since the latter stages need to discriminate better between positive and negative samples, *feature extraction algorithms* may be used to improve

accuracy, which however, further increases computational demands. In such cases, it is possible to use the reduced-set-method [14], to reduce the number of support vectors required by the non-linear kernel stages in order to improve classification times.

2.3 Object Detection

The process of visual object detection deals with determining whether an object of interest is present in an image/video frame or not and also determine its location within the frame. The overall visual object detection process begins by first receiving an input image/video frame from a camera or other adequate image source, which subsequently will then be searched in order to find possible objects of interest. This search is done by extracting smaller regions from the frame, called *search windows*, of $m \times n$ pixels, which are processed by a classification algorithm to determine if they belong to the object of interest class or not [4]. Thus, the classification algorithm, such an SVM, learns to categorize search windows of a particular size. However, the object of interest may appear in the image/video frame at a larger size than the size of the search window. In such a case, the classification algorithm will not be able to detect the object. To account for this scenario an object detection system typically decreases the size of the input image (downscaling), effectively reducing the size of the object of interest, and then reexamines the downscaled image with the same search window size. The downscaling process is done in steps to account for various object sizes, down to the size of the search window and scaling happens by mapping old coordinates to new ones using a $scalingfactor$. Hence, many downscaled images are produced from a single input image/video frame, each in turn produce a number of search windows, which increases the amount of data that must be processed by the classification algorithm such an SVM. Search windows can be extracted from every pixel location in the image (exhaustively) or every few pixels and is called the *window pixel step*. This search process is what generates a large number of windows that need to be classified. Also considering that a substantial number of those windows are background and not an object of interest, it becomes apparent that visual object detection is a prime example of an application that can benefit from the cascade classification structure.

It is important to consider the metrics used to measure the performance of an object detection system. An image object detection system is characterized by how accurately it can classify data as well as how many image frames it can process per second. Thus, the two commonly used performance metrics are the *detection accuracy*, and *frames per second* (FPS) or *frame rate*. Detection accuracy is usually measured on a given *test set* where the expected outcome for a sample is compared to the actual outcome of the object detection system. The detection accuracy is the percentage of samples for which the expected outcome matches the actual outcome of the detection system. FPS concerns the throughput of a system and is the maximum number of digital video/image frames, of a given size, that the detection system can process in one second. A performance of $30FPS$ is usually considered a minimum in order for an object detection system to be capable for *real-time* video processing.

2.4 Local Binary Patterns (LBP)

Each window that is extracted from the image is processed to produce features that provide invariance to different lighting conditions and other environmental variations. These features can either be shape, color, intensity, or the result of various filters and feature extraction algorithms. Using features makes the detection process more robust since it provides a more representative description of the object and reduces the within-object-class variability. However, the addition of feature extraction approaches and preprocessing methods can have a negative effect on the classification speed even though the accuracy can be improved. In this paper we use the features called Local Binary Patterns (LBPs) [17], that describe the relationship between a pixel and its neighborhood, and have been used in a wide range of computer vision applications [18,19]. The major advantage of LBPs is their low computational complexity while providing sufficient accuracy which makes them particularly attractive for embedded applications where the available resources may be limited and low power operation is needed. Many variants of LBPs exist in the literature that are tailored to different objects [19]. The approaches that are of interest in this work are the standard LBP [19] and Center-Symmetric LBP (CS-LBP) variation, The former has been used for face detection and recognition while the latter for pedestrian detection. The process of extracting the LBP features first begins by thresholding a 3×3 neighbourhood (image sliding window). For the standard LBP (Fig. 2a) the values are threshold with the center value of that neighborhood placing 1 where the value is greater or equal than the mean, and 0 otherwise. For the CS-LBP the difference of opposite direction pixels, symmetrically to the center, is compared against a predetermined threshold and if it is grater the resulting value is 1 and 0 otherwise. The produced binary map is multiplied with a predefined mask (usually incremental powers of two). The values are then summed to obtain the LBP Code. The result of LBP processing is an image assembled by LBP features (Fig. 2b). In order to improve the robustness of LBP features it is suggested [19] that the LBP code is further processed to compute rotation invariant features. This is done by characterizing the code as uniform and non-uniform based on the bit transitions in the code. The number of transitions in the code is found next by *xor* operations on the LBP code bits. This is necessary to identify a pattern as uniform LBP code (which has 2 or less transitions e.g. 11110000) or non-uniform LBP code (which have more than 2 transitions e.g. 10100101). This offers a more meaningful interpretation of the LBP codes which can achieve higher discrimination. The next and final step to creating the LBP-based descriptor requires dividing the LBP-based in k blocks of $i \times j$ pixels. A local histogram is generated for each block in the image.

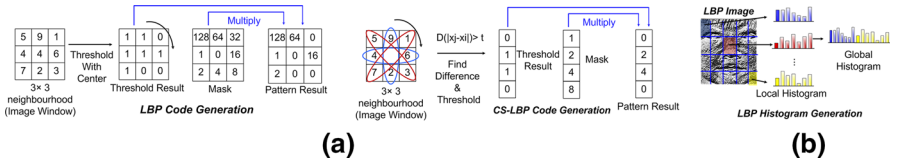

Fig. 2 Local binary patterns: **a** LBP Code generation using improved LBP and Center Symmetric LBP. **b** LBP histogram processing

The local histograms can have any number of bins depending on the application. The local histograms are then concatenated to form a single global histogram descriptor, as shown in Fig. 2b.

2.5 Related Work

Software implementations of cascade SVM classification schemes [6,7,14,15,20,21], have shown speedups over monolithic SVMs and although noteworthy and suitable for some applications, are yet to offer adequate performance for real-time resource-constrained applications. This is due to the fact that the latter stages become the bottleneck since they require processing an increased number of SVs and the requirement for parallel processing arises. Hence, hardware accelerators based on parallel computing platforms (e.g. [22,23]) for SVM classification have been proposed in the literature 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. A brief description of related works follows next.

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 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 are not available for a full parallel processing 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 ($\sim$100 elements) [9,24,25] and small-scale multiclass implementations [26] in order to be able to meet real-time constraints. In addition, these architectures cannot trade-off processing more SVs rather than vector elements, and hence, cannot efficiently deal with the different computational demands of the cascade SVM stages. The works in [10,27] target image recognition problems for 32×32 resolution images. However, the datasets consist of only a few images corresponding to low vector dimensionality. Barcode detection is the targeted application for [28] operating on 512×512 resolution images and scanning window size of 16×16 (256 element vector) and requires 352 cycles to classify an input vector. Alternative approaches include FPGA coprocessors for parallel vector processing in order to speedup SVM computations [8,29]. 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 PCI express and external DRAM which have high power consumption and are thus unsuitable for embedded applications.

In addition, research has also been done on potential simplifications to make SVM classification more hardware friendly and improve its suitability for devices with limited computational resources. These approaches include using CORDIC algorithms to compute the kernel functions [10,24,27,30,31]. However, the iterative operations

of these algorithms make it challenging to achieve high performance for applications that require high data throughput such as object detection, since compact implementations of CORDIC algorithms which require less hardware, have increased latency [32]. Other works [33,34] proposed that the computations be done in the logarithmic number system so that all multiplications are substituted by additions, thus reducing computational 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. Alternatively, a pseudo-logarithmic number system was proposed in [35], however, the overhead for converting between number systems, in order to perform additions, remains. The works in [36–38] have looked at how the bitwidth precision impacts the classification error, in an effort to find the best trade-off between hardware resources, performance and classification speed.

Parallel computing using GPUs has been increasingly used in recent years in order to speedup SVM classification by taking advantage of the parallel processing capabilities of a GPU showing improved results compared to CPU implementations [20,39]. A hybrid FPGA-GPU pedestrian detection is presented in [40] where the SVM is implemented on the GPU and a feature extraction algorithm on the FPGA for 800×600 images and achieves over 10 frames-per-second for the classification of 1000 windows. However, GPUs are power hungry devices compared to FPGAs [29,41], (FPGAs consume approximately an order of magnitude less power as shown in [12]) and as such they are not suitable for power-constrained embedded applications. In addition, existing GPU implementations do not translate well to the more energy-efficient embedded GPUs due to less available resources [42].

It is evident from the above that there is limited work for the hardware implementation of cascade SVMs as most of the related works consider only monolithic SVM classifiers. Hence, efficient ways to utilize the different computational demands of cascade SVMs stages have not been sufficiently examined. Only recently there has been some work in the hardware implementation of cascade SVM classifiers [13,41]. Moving towards large scale embedded applications and problems where thousands of samples need to be classified, 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.

3 Proposed Design Method and Hardware Architecture

Motivated by the aforementioned discussion and the need to consider efficient hardware architectures for cascade SVM classifiers we propose parallel hybrid hardware architecture, with two main processing modules that offer different parallelism with respect to the SVs, to provide higher classification throughput and a hardware reduction method leading to a more compact hardware implementation suitable for embedded system applications. We also show how a suitable I/O structure can be designed to facilitate data input to each processing module. In addition, the architecture also incor-

porates a novel compact feature extraction processor based on local binary pattern (LBP) descriptors [19], targeting object detection applications.

3.1 Cascade SVM Hardware Reduction Method

Existing cascade SVM classification schemes utilize a hierarchy of SVM classifiers which can be different classifiers or expansions of a single classifier. Nonetheless, the common feature of these cascade structures is that stages at the beginning of the cascade usually require processing less SVs than subsequent stages making them computationally less demanding. This is because the objective of the early SVM stages 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. However, this implies that a few negative samples will be classified as positive. In contrast, subsequent stages need to be more accurate and discriminate better between positive and negative samples; and hence build decision functions with more SVs.

The fact that the early SVMs stages are not optimal classifiers can be exploited to reduce the resources required 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 approximate 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. To adjust the accuracy of each cascade stage rounded off to the nearest power of two, to similar rates of that of the initial cascade stages the receiver-operating-characteristic (ROC) curve is used. The ROC curve shows the performance of a binary classification system by illustrating the corresponding true positive and false positive rates, as the discrimination threshold is varied. As such, by setting the appropriate threshold the performance of the adapted stages in the SVM cascade can be adjusted to match the true positive rate of the initial SVM cascade stages. This is important since the true positive rate of the classifier must be maintained. Adapted stages, which do not yield the targeted accuracy, are reverted back to the initial model. The process is summarized in Fig. 3. The hardware reduction process takes place after the cascade structure is decided, meaning that the kernel function, and number 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 depend on the specific hardware architecture used for the implementation of the cascade and as such can be optimized to different architecture requirements.

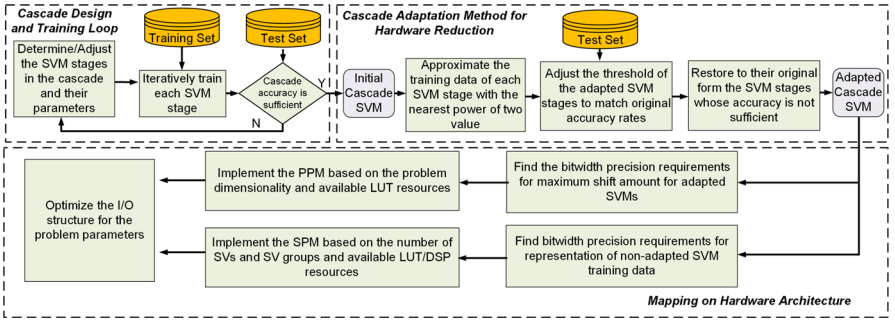

Fig. 3 Cascade hardware reduction method: the initial cascade SVM obtained after training is altered to become more hardware friendly resulting in an adapted cascade SVM

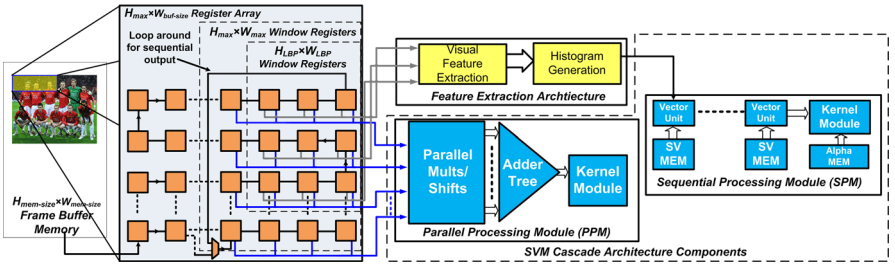

Fig. 4 SVM cascade system architecture comprised of the sequential processing module (SPM), the parallel processing module (PPM), the register array, frame buffer memory, and the visual feature processor

3.2 Hybrid Cascade Hardware Architecture

The proposed architecture (Fig. 4) consists of the main cascade processing components as well as additional components, which relate to the targeted benchmark application of object detection [5]. The presented architecture is flexible, and the parameters of each component can be adjusted to meet the requirements of the given classification problem. Furthermore, the modular design means that the architecture can support different processor modules, which makes it suitable for different cascade implementations. Due to the nature of the cascade classification scheme, each successive SVM stage will have fewer input data to process and more SVs to process than the previous stage. Hence, efficient hardware architectures need to take into consideration 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 stages. The first is a fully parallel processing module (PPM) which performs the processing necessary for all the adapted SVM stages. The PPM is used to process a single vector in parallel and can be designed to process vectors of different dimensionalities and can be unrolled to be as parallel as the available resources allow. The second is a sequential processing module (SPM), optimized for the high complexity SVM stages which demand processing a large number of SVs but only a fraction of the input data. Thus parallelism focuses on

processing multiple SVs in parallel. The vectors can have different dimensionality and the number of parallel SVs that can be processed can also be adapted to the available amount of resources. In addition, a specialized processor for LBP histogram extraction is implemented with a low-resource consuming architecture as a means to efficiently improve accuracy.

3.2.1 Parallel Processing Module (PPM)

The parallel processing module (PPM) carries out the processing of the low complexity SVM stages which have been adapted using the previously described hardware reduction method. Specifically, the proposed architecture can process linear and 2nd 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 [43]. The characteristic of the early cascade stages is that they require processing 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 with a single SV to reduce the processing time per vector.

The architecture of the PPM (Fig. 5) is designed to perform the SVM shift operations, accumulations via an adder tree pipeline, and the final kernel computations. As such, it is comprised of parallel SV data memories, arithmetic shifters and parallel sign conversion units. In addition, it is also 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 components are dedicated to kernel processing and are also mostly implemented using arithmetic shift units. The shift data are fetched in parallel from small ROMs, and include the sign of the support vector, the shift amount, and the direction of the shift operation. The parallel processing module starts by first processing the input vector elements with a sign conversion unit which that converts the input to positive or negative according to the SV sign bit so that the initial multiplication operation can be preserved. The signed numbers are then processed by arithmetic shift units

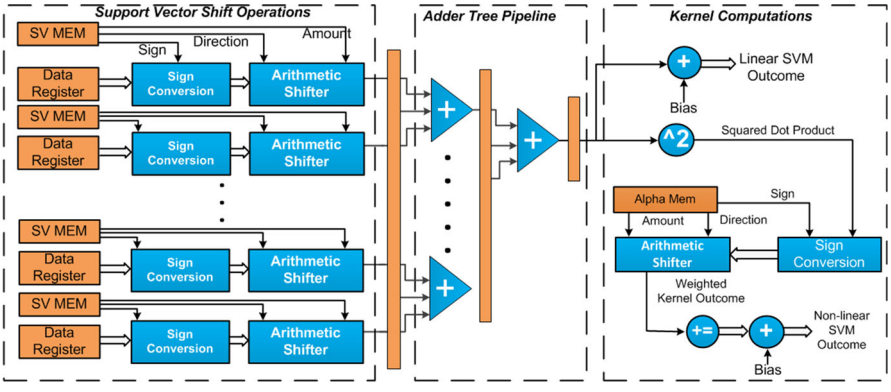

Fig. 5 Parallel processing module (PPM) architecture: Handles the processing of the nearest power of two adapted SVM stages. The shift units and adder tree are used by all kernels while only non-linear kernels use the rest of the kernel module

which perform the shift according to the data that they receive from the ROMs, while preserving the sign bit, and correspond to the multiplication operation with the SV. The partial results are added together using a pipelined tree of adders so that the dot-product outcome can be obtained. The length 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. The latency of the adder tree is thus given by:

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

When 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 2nd degree polynomial kernels (3), as well as other kernels [e.g. (4)] additional operations are necessary. The kernel computation module handles the latter steps of the classification phase. A single multiplier is used in the parallel processing module and is necessary to perform the squaring operation of kernel (3). The alpha coefficients are also approximated with power of two values, and hence, their processing is done with a sign conversion unit and an arithmetic shift unit similarly to the processing of the SVs. An accumulator is used to gather and process 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 Eq. (6), where $N_S(i)$ is the number of support vectors that need to be processed by stage i.

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

3.2.2 Sequential Processing Module

The processing of the latter SVM stages (typically the final SVM stage) is performed via the sequential processing module (SPM). This final stage will most likely process only a small percentage of the input data; however, it will have the largest number of SVs. Hence, a different architecture is needed for such purposes, instead of the PPM, which needs to be compact and also offer parallel processing. Using the SPM architecture allows us to process multiple SVs with a single vector element at a time facilitating both parallel processing and a more compact and modular design. Therefore, 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. 6, which is comprised of a series of pipelined processing and memory units. The majority of the units in the module are vector processing units (VUs) 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, along with register and multiplexer logic for data transfer between vector units. The last 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.

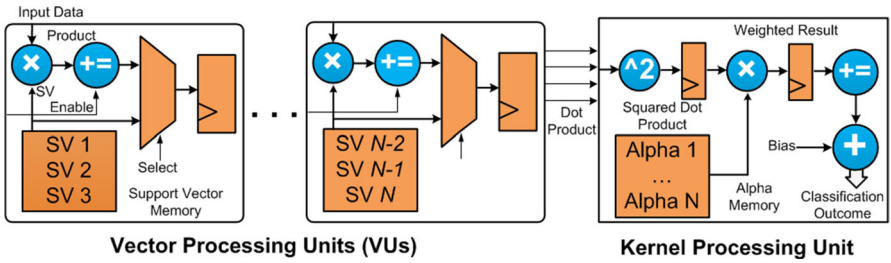

Fig. 6 Sequential processing module (SPM) architecture: Consists of two processing units: The dot-product processing units handle the dot-product computation, and the kernel processing unit, which is shared amongst the dot-product units, handles the kernel-related operations

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. When processing more than one group of SVs which happens in cases when the processing resources are not enough to process all SVs, then each group is processed after the other. In total depending on the number of groups a total of $\lceil N_{SV}/num\_of\_VUs \rceil$ processing repetitions are necessary. Hence, the size of the pipeline can be adjusted to fit the available resources and processing requirements by fine-tuning the number of support vector groups. Each vector processing unit in the pipeline processes one element of the support vector with one element of the input vector at a time. The data in the SPM flows in a pipelined manner as the input vector values are propagated from one unit to the next, through the dedicated transfer mechanisms; while the ROMs feed each VU with SV data in parallel. 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 2 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 Eq. (7).

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

3.3 Local Binary Processor Architecture

The LBP processor will only selectively be used by the latter cascade stages to improve the discrimination capabilities for a small number of samples that exhibit common features but belong to different classes. Hence, it must have low area overhead. This is

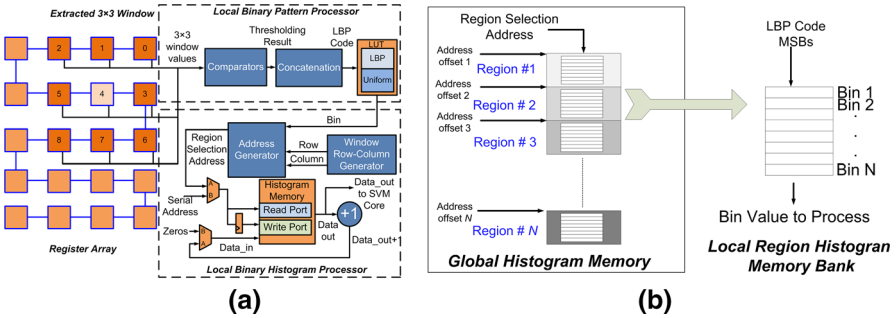

Fig. 7 **a** Local binary pattern processor architecture, **b** histogram update process

different to works such as [44] where the goal is to parallelize the LBP processing. As such, a fully parallel implementation that may consume substantial hardware resources is not suitable. Through efficient design, we are able to develop a novel and compact LBP processor suitable for the targeted applications (Fig. 7a). The developed LBP processor architecture features parallel processing of the values of only a single 3×3 window from the input image located in the register array. In addition, the LBP processing begins from the second row and column, i.e. the image boundary is ignored, hence, no additional hardware was necessary to pad the image with additional values. The processor is essentially comprised of two main blocks. In the first block, which is the LBP code generation, the values corresponding to a 3×3 window are loaded from the register array and processed accordingly either for CS-LBP or standard LBP features, as discussed in Sect. 2.4, to produce a binary array which is then multiplied with a power of two mask. Effectively, however, the multiplication with the mask can be implemented by simply concatenating the threshold bits to generate the LBP code. Once a window is processed the next 3×3 window follows the next cycle. The LBP code is then processed to characterize it as uniform or non-uniform. We employ a look-up table memory approach in order to achieve a fast processing of the LBP code. The code is used as the address to the memory and the output is the number of transitions in the code.

The second major block is the histogram generation circuit which receives the processed LBP codes for each 3×3 window and computes the histogram descriptors. A major issue when dealing with histogram generation is that of collisions to the histogram memory from increments to the same location. The architecture generates and processes a single LBP code every clock cycle, hence it is collision-free which results in a much simpler implementation. The actual histogram computation is carried out in two phases. As the histogram is stored in a central memory (of size $k \times i \times j$) which contains all local histograms; the first phase is to find the starting address for the local histogram which the LBP code belongs to (Fig. 7b). This is achieved by counting the row and column of each LBP code. By keeping track of the MSBs of the row and column coordinates it is possible to identify the block which it belongs to. Then by setting the appropriate address offset the corresponding histogram region is selected. From there the desired most significant bits (MSBs) of the LBP code are added to the address offset in order to select the appropriate bin (memory location)

where the LBP code belongs to in the local histogram, and increment its value (Fig. 7c). A dual ported memory is utilized to store the histogram. In this way, an immediate reset can be performed right after a memory value is loaded to the SVM processing core from the second port, which receives the same address but delayed by a single cycle. Using this dual-ported memory scheme leads to a more simple implementation, without needing to use a pool of registers, multiplexers, and additional wiring. Overall, the LBP feature processing unit is generic and can thus deal with processing different image sizes, number of blocks, and different number of histogram bins.

3.4 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. It should also take advantage of the application-specific characteristics to facilitate data reuse and reduce memory accesses. Furthermore, different classifiers may utilize different data points or need to preprocess the data. The cascade I/O structure should be able to handle this. 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 register array and are propagated column-wise within the structure. The register array has a size of size $H_{max} \times W_{buf-size}$, where H_{max} is the height of the maximum window and $W_{buf-size}$ 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 column-wise into the structure. The image region that is at the right-most part of the register array corresponds to a $H_{max} \times W_{max}$ window and each unit receives data from specific registers that window. Specifically, the LBP processor receives 9 pixels from the right-most 3×3 window ($H_{LBP} = 3$, $W_{LBP} = 3$) to produce a $H_{max} - 1 \times W_{max} - 1$ image made up of LBP codes which is later processed in $ki \times j$ blocks to produce a histogram descriptor. The PPM can receive register data corresponding to either a $H_{max} \times W_{max}$ window or any other downscaled version [e.g. a $(H_{max}/2) \times (W_{max}/2)$ window] if it is necessary, by selecting the appropriate registers, thus achieving dynamic downscaling of the larger $H_{max} \times W_{max}$ window. With 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 leftmost region of the scanline buffer and is ready to be processed next. The data flow of the right-most registers changes depending on whether the data are used for parallel or sequential processing. In the case of parallel processing module, 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 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 (Fig. 3). This is required so that

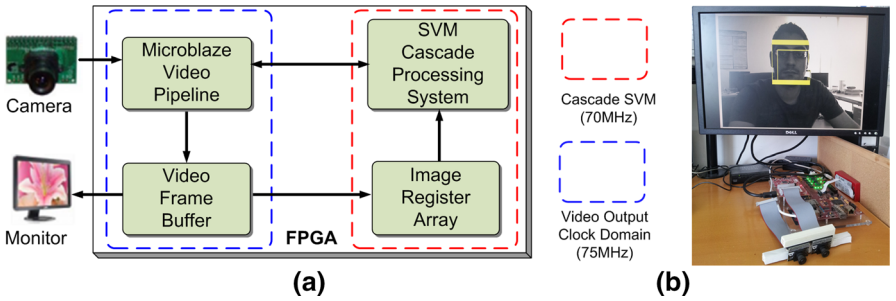

Fig. 8 **a** Block diagram of the FPGA system components. **b** Xilinx Spartan 6 FPGA industrial video processing board and setup for Face Detection Implementation

the window is placed correctly with respect to the rest of the image in the register array in order to maintain consistency. Moreover, this is necessary so that the same window can be processed again with a new group of support vectors when needed. The register array structure, which stores the part of the image to be processed, can be implemented with different number of rows and columns to fit the desired image size given the available hardware resources.

4 Experimental Platform and Results

The proposed hybrid hardware architecture and methods were evaluated using the embedded applications of face and pedestrian detections considering 800×600 (SVGA) resolution images. For both applications the architecture was evaluated in terms of frame-rate, detection accuracy, power consumption, as well as requirements in terms of computing resources. The cascade structure, was trained using MATLAB and was used to evaluate the hardware architecture and proposed design methodologies. Additionally, 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, but without applying the hardware reduction method, and thus the parallel processing module is implemented using multipliers and not shift units. 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 [45] video-pipeline 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 between balancing I/O delays and hardware resources. The following sections detail the evaluation process and the results.

4.1 SVM Cascade Structure and Training

To evaluate the proposed hardware architecture and approaches we designed an SVM cascade structure for each application, with kernels, and parameters similar to what has been used in the literature [5–7]. The parameters for each application are summarized

in Table 1. The early stages tasked with the fast rejection of samples correspond to linear SVMs, or non-linear kernels with low computational demands [5]. Hence, for both applications the cascade structure is similar to the one depicted in Fig. 9. Specifically, for the face detection application, the cascade structure is comprised of two linear and two polynomial non-linear SVMs The search window size is 20×20 pixels and are extracted every 5 pixels, which is similar to other works of cascade SVMs in the literature [5]. This window size results in a 400-dimensional vector, which is provided to the first three SVM cascade stages for rapid processing. Once a window has passed all three stages successfully, it is further processed to produce an 18×18 LBP feature image which corresponds to a 1062-dimensional LBP histogram vector which in turn is passed as input to the final SVM cascade stage. Similarly, for the pedestrian detection application, the cascade structure was comprised of 4 linear SVM stages and 1 non-linear polynomial SVM stage. This structure processed 36×18 pixel windows extracted every 5 pixels resulting in a 648-dimensional vector. Prior to the final stage, the windows that passed all previous stages, were processed by the LBP processor that produced a 1512-dimensional vector, which was propagated to the final stage.

Positive and negative samples for both applications were collected from commonly used databases [46,47], and used to setup an initial training set which was then enhanced with additional samples. The early cascade stages were trained in incremental fashion [1–3]. The first stage was trained on the initial training set and adapted using the hardware reduction method. Then, the initial training set was enhanced with negative samples that were misclassified by the first stage, and the new training set was used to train the second classifier, and the same was done for the next stages except the last one. 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. Consequently, a 80 SV polynomial SVM was generated for face detection and a 100 SV polynomial SVM was generated for pedestrian detection. The adapted stages retained similar accuracy level after being rounded-off to the nearest power of two and hence where implemented on the PPM. However, for the final SVMs there was a significant discrepancy between the classification accuracies of the adapted and original model, since it approximates a more complex decision function. Hence, those were not approximated and were implemented on the SPM.

4.2 FPGA Implementation and Logic Resource Utilization

The two cascade implementations (baseline and adapted) have the same basic architecture (Fig. 3) and data flow for both applications. The PPM architecture was based on a fully unrolled implementation, while the SPM was implemented with a number of DSP units equal to the number of SVs ($num\_of\_VUs = N_S$) meaning that the input data to the SPM is processed only once with a single group of SVs. The only difference between the two implementations is that in the adapted cascade case the PPM 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 corresponded to shift values instead of support vector values. Each ROM holds the support vector data for the first three cascade SVM stages for the specific vector

Table 1 Parameters for each application

Application	Image resolution	Search window size ($H_{max} \times W_{max}$)	Downsampling rate	Window step	LBP block size	Number of LBP blocks	LBP histogram bins	Number of windows	Cascade SVM stages	Cascade SVM stages	Total SVs
Face detection	800 × 600	20 × 20	1.2 (18 scales)	5 pixels	i=3, j=6	k=18	l=59	56,894	4	2 linear 2 non-linear	122
Pedestrian detection		38 × 16			i=3, j=2	k=108	l=4	54,008	5	4 linear 1 non-linear	104

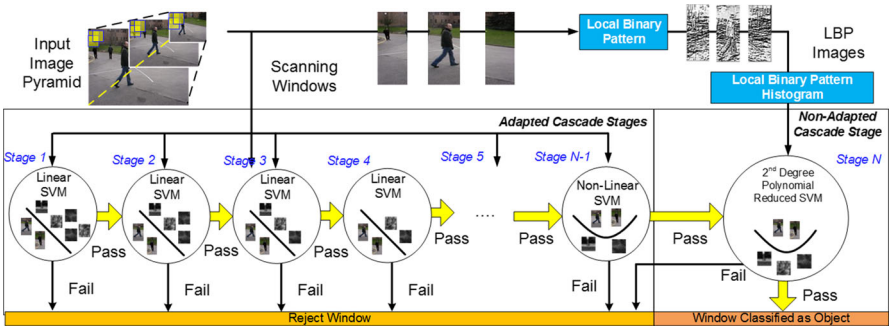

Fig. 9 Cascade hardware reduction method: The parameters of the initial cascade SVM obtained after training are changed to produce a more hardware friendly adapted cascade SVM

Table 2 FPGA resource utilization breakdown per unit and system for face detection system

FPGAResources	LUTs (92152)	Registers (184304)	BRAMs (268)	DSPs (180)
SPM	4482 (4%)	3472 (2%)	51 (19%)	50 (27%)
Adapted PPM	19,006 (20%)	2679 (2%)	1 (< 1%)	—
Baseline PPM	30,791 (46%)	3724 (3%)		
Generic LBP processor	150 (1%)	40 (< 1%)	2 (< 1%)	—
Memory & I/O Units	1200 (1%)	1831 (1%)	180 (67%)	—
Microblaze video pipeline	9891 (10%)	10,780 (5%)	20 (7%)	3 (2%)
Baseline cascade system	47,396 (51%)	21,214 (11%)	256 (96%)	89 (32%)
Adapted cascade system	35,532 (38%)	20,153 (11%)		

elements. In the adapted cascade implementation, 6 bits are needed to store the shift data: 4 bits for the shift amount, corresponding to a maximum shift amount of 15 bits, one bit for the sign of the support vector, and one for the arithmetic shift direction. For the baseline implementation, 8 bits are needed to represent the decimal number SVs to maintain the same accuracy. In addition, adder trees, used by the PPM, utilize ternary adders instead of two-input adders, to reduce the latency.

Both implementations on the Xilinx Spartan-6 XC6SLX150T FPGA have the same critical path, which is the SPM kernel unit mapped on the DSPs, and as such have the same operating frequency of 70 MHz. The results in Table 2 provide a more detailed breakdown of the the utilized resources for each component for the face detection system, while the same trend applies for the pedestrian detection system. Specifications and resources for the complete system for both applications are shown later in Table 3. Overall, for both the face and pedestrian detection applications the implementation of the adapted PPM requires ∼40% fewer FPGA logic resources compared to the baseline PPM. This is reflected with a 25full system implementations.

4.3 Detection Accuracy and Frame-Rate

Accuracy and frame-rate are two important metrics in object detection and thus this section outlines these results. The accuracy of the adapted cascade SVM was evaluated

Table 3 Comparison of related work on FPGA-based implementations of SVM detection systems

Related works	Rojas [28] [a]	Kryjak [44]	Bauer [40] [b]	Presented work	
Application	Barcode detection	Head detection	Pedestrian detection	Face detection	Pedestrian detection
Method	Polynomial SVM	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 [b]	Xilinx Spartan 6 XC6SLX150T	
FPGA resource					
LUT	22,938/28,672	12,068/150,720	28,616/62,208 [c]	35,532/92,152	45,758/92,152
REG	N/P	15,893/301,440	N/P [c]	20,153/184,304	25,049/184,304
BRAM	160 KB	124/416	100 [c]	256/268	260/268
DSP	N/P	66/768	18/96 [c]	89/180	109/180
Image size	512 × 512	640 × 480	800 × 600	800 × 600	
Window size	16 × 16	32 × 24	48 × 96	20 × 20	36 × 18
Vector sizes	256	1440	1980	400 & 1062	648 & 1512
Total number of SVs	88	1	N/P	102	104
Frequency	166 MHz	120 MHz	63 MHz [c]	70 MHz	
Detection	TP: 91,8	TP: 83%	TP: 95.4%	TP: ~ 81%	TP: ~84%
Accuracy	FP: 4,2%		FP: 0,1%	FP: ~0,03%	FP: ~0,07%
Detection speed	N/P [a]	60 FPS	10 FPS	36 FPS	34 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. *N/A* not applicable, *TP* true positive, *FP* false positive

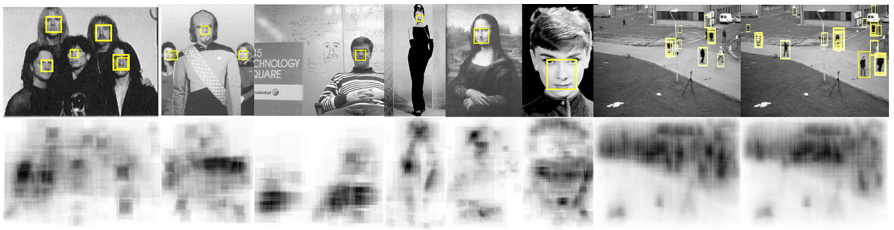

Fig. 10 (*top*) Detection results on 800 × 600 images for face detection [49] and pedestrian detection [50]. (*bottom*) Activity in the image. The *darkness* of the pixels indicates that the image region has gone through more stages

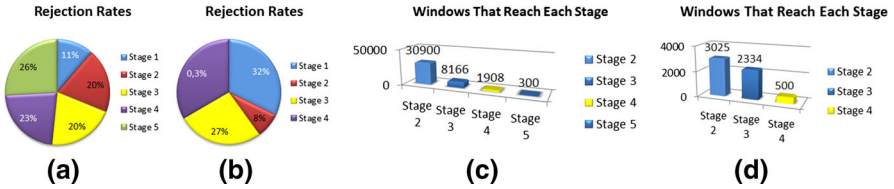

Fig. 11 Window rejection rates for each stage for **a** pedestrian detection and **b** face detection. Number of windows that reach each stage for **c** pedestrian detection and **d** face detection

on the widely used face image databases [48,49] as well as pedestrian databases [50] and images from the world-wide-web, cropped and resized to 800 × 600 (SVGA) resolution. Full frame detection results are shown in Fig. 10. The same set was used to evaluate the frame-rate of the cascade SVM implementations. Each 800 × 600 image generates a total of over 54,000 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 generated windows are processed by the first SVM stage; however, only ∼1% of them reach the final SVM stage. This is shown in Fig. 11, which illustrates the percentage of windows rejected at each stage and the number of windows that each stage, after the first one, needs to process. In addition, to the actual processing time, the I/O delays per frame also negatively affect the classification speed. In order to achieve higher detection rates, I/O and memory operations such as filling the register-array buffers, overlap with window processing. Overall, the implemented architectures were able to operate in real-time achieving of 36 FPS for face and 34 FPS pedestrian detection respectively. In addition, the detection accuracies for each implemented hardware architecture were ∼81% for face detection and ∼84% for pedestrian detection respectively, while for both cases a considerably low false positive rate was achieved (<0.1%).

4.4 Trade-Off between Resource Utilization and Accuracy

The penalty to pay for any form of approximation is that the accuracy changes compared to the original model. However, the main benefit from the proposed hardware reduction method of approximating the SVs and alpha coefficients by rounding them

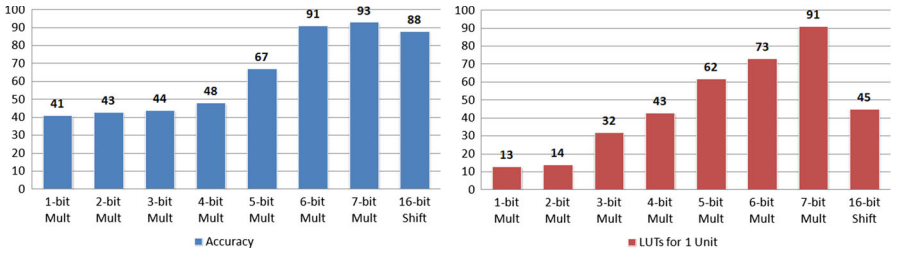

Fig. 12 Trade-off between accuracy and resources

to the nearest power of two is that it allows for a more compact and parallel implementation without a significant impact on accuracy. To better understand the benefits consider the alternative which is the conventional approach where the coefficients are approximated by a fixed point number of a certain bitwidth selected in such a way as to retain a similar accuracy to the original model and in which case the main processing units are multipliers. We illustrate in Fig. 12 for one of the SVMs in the cascade for the face detection application, the accuracy that we get and the LUT resources necessary for one multiplication unit as the bitwidth that we use to represent the SVs and alpha coefficients increases. We also illustrate (in the last column of each figure) the accuracy of the equivalent SVM approximated with power of two values and the LUT resources for one shift unit. In order to maintain an accuracy of over 90% for the conventionally adapted SVM we need 6 bits which results in a multiplication unit requiring 73 FPGA LUTs. Conversely, a shifter for the SVM with power of two values requires 46 FPGA LUTs with a slightly reduced accuracy but still approaching 90%. Maintaining a similar number of LUTs using a multiplier resulted in a very low accuracy of 48%. It is evident that increasing the number of bits increases accuracy as we get closer to the actual values, however, the required resources for the multiplication units also increase. Especially when considering parallel implementations where the increased number of LUTs needed by the multipliers accumulates and results in a resource hungry implementation. However, through the proposed cascade design approach we are able to achieve an adequate trade-off and reduce the hardware implementation requirements without sacrificing accuracy.

4.5 Power Consumption

Power analysis tools from Xilinx were used to measure power consumption figures of the adapted and baseline cascade SVM FPGA implementations for both benchmark applications. 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 video pipeline, for the adapted cascade SVM system ranged from 4.4 to 8.0 W for face detection and 4.8 to 8.6 W for pedestrian detection. While for the baseline cascade system, it ranged from 4.4 to 9.9 W for face detection and 4.8 to 10.4 W for pedestrian detection. As the peak power consumption

happens when the PPM module is used, the utilization of less LUT resources by the adapted PPM results in reducing the peak power needed for the adapted cascade system by ∼20%.

4.6 Related Work Comparison

Related works for object detection applications are shown in Table 3 along with information regarding parameters and performance. These works use different algorithms, training and test sets, and benchmark applications; hence, it is difficult to make a direct comparison between implementations. In addition, none of the reported works target cascade SVM processing, hence, the proposed architectures are not optimized to dynamically handle different SVM processing requirements. Regardless, through the subsequent discussion we attempt to highlight the contributions of this work compared to what is reported in the literature. SVMs have been used in various object detection applications and as a result FPGA implementations for SVM-based object detection have used different applications and parameters to benchmark the proposed architectures. However, since the SVM classification flow treats all data as vectors; the number of samples and SVs processed and vector dimensionality can provide an indication to the processing performance for each work. The number of samples depends on the search window size and granularity of the search. Because of the different benchmark applications, the search window size and feature vector size are different.

A head-shoulder detection system is presented in [44]. It utilizes a linear SVM and LBP descriptors to classify 19,200 windows from 640 × 480 images. It trades-off accuracy for performance by 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. However, non-linear kernels often provide better and more robust results compared to linear kernels and thus might be the preferred choice in many applications in which case such architectures will not be able to handle the different processing requirements. The implementation in [28] scans a 512 × 512 image in non-overlapping 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. Furthermore, it processes only around 1024 16 × 16 window samples, corresponding to 256-dimensional vectors, per image, and it does not downscale the input image which simplifies the I/O and memory accesses. The hybrid FPGA-GPU pedestrian detection system [40] 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 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 (800 × 600) which generate more windows (over 54,000), with a higher number of SVs (one to two orders of magnitude more) while also reducing the implementation requirements for the implementation of

more complex cascade SVM classifiers. The SVM hardware implementations target different applications and thus accuracy is difficult to compare. On the other hand, software based implementations [5–7] that utilize cascade SVMs for face detection achieve accuracies that range between 78–80% while utilizing similar training set sizes. The proposed optimized SVM cascade system achieves a detection rate of over 80%, which is on par with these works.

5 Discussion and Concluding Remarks

Overall, the proposed architecture enhanced with the proposed methods achieves a good trade-off between accuracy, performance, and hardware utilization. In addition, there are some useful conclusions extracted from the FPGA evaluation. Firstly, the proposed hardware reduction method provides an efficient way to reduce the required silicon budget of the cascade which is easy to implement and compatible with any SVM training package and hence, can have a wide use. As such, it can be used to design low-cost SVM pre-processors that can be integrated to other existing works as well. Second, the hybrid processing hardware architecture and flexible I/O structure demonstrated how it is possible to efficiently utilize the available hardware and provide the necessary performance; by optimizing the design for the specific data flow, throughput and processing demands of each cascade stage.

This work presented in this paper demonstrated how application directed design optimizations can help boost the hardware efficiency of cascade SVM implementations so that such structures can be used to design intelligent embedded classification systems. The proposed hardware architecture and design methods were verified experimentally on a Spartan-6 FPGA using the applications of face and pedestrian detection. Through the evaluation we showed the effectiveness of the proposed architecture as it was capable of processing 800×600 resolution images with a real-time performance of over 30 frames per second for both benchmark applications. In addition, the proposed hardware reduction method resulted in reducing the required custom logic resources by $\sim$30%. Going forward different applications will also be used to evaluate the proposed architecture to further demonstrate how it can be used to provide a complete embedded solution for the design of intelligent classification systems.

Acknowledgements This work was supported in part by the ERC Advanced Grant "Fault-Adaptive", ERC grant agreement no 291508.

References

1. Cortes, C., Vapnik, V.: Support-vector networks. J. Mach. Learn. **20**(3), 273–297 (1995)
2. Burges, C.J.C.: A tutorial on support vector machines for pattern recognition. Data Min. Knowl. Discov. **2**, 121–167 (1998)
3. Osuna, E., Freund, R., Firosi, F.: Training support vector machines: an application to face detection. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 130–136 (1997)
4. Wu, J., Rehg, J.M.: Chapter 8 object detection. In: Ensemble Machine Learning: Methods and Applications, 1st ed., pp. 225–250. Springer-Verlag, New York (2012)

5. Heisele, B., Serre, T., Prentice, S., Poggio, T.: Hierarchical classification and feature reduction for fast face detection with support vector machines. Pattern Recognit. **36**(9), 2007–2017 (2003)
6. Kukenys, I., McCane, B.: Classifier cascades for support vector machines. In: International Conference on Image and Vision Computing, pp. 1–6 (2008)
7. Ma, Y., Ding, X.: Face Detection Based on Cost-Sensitive Support Vector Machines. In: First International Workshop on Pattern Recognition with Support Vector Machines, pp. 260–267 (2002)
8. Cadambi, S., et al.: A Massively Parallel FPGA-Based Coprocessor for Support Vector Machines. In: IEEE International Symposium on Field Programmable Custom Computing Machines (FCCM), pp. 115–122 (2009)
9. Pina-Ramirez, O., Valdes-Cristerna, R., Yanez-Suarez, O.: An FPGA implementation of linear kernel support vector machines. In: IEEE International Conference on Reconfigurable Computing and FPGA's, pp. 1–6 (2006)
10. Ruiz-Llata, M., Guarnizo, G., Yébenes-Calvino, M.: FPGA implementation of a support vector machine for classification and regression. In: International Conference on Neural Networks, pp. 1–5 (2010)
11. Fowers, J., Brown G., Cooke, P., Stitt, G.: A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications. In: ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '12), pp. 47–56 (2012)
12. Cooke, P., Fowers, J., Brown, G., Stitt G.: A Tradeoff Analysis of FPGAs, GPUs, and Multicores for Sliding-Window Applications. In: ACM Transactions on Reconfigurable Technology Systems, vol. 8, no. 1 (2015)
13. Kyrkou, C., Bouganis, C.-S., Theocharides, T.: An Embedded Hardware-Efficient Architecture for Real-Time Cascade Support Vector Machine Classification. In: International Conference on Embedded Computer Systems (SAMOS), pp. 129–136 (2013)
14. Burges, C.J.C.: Simplified support vector decision rules. In: International Conference on Machine Learning, pp. 71–77 (1996)
15. Sahbi, H., Geman, D., Boujemaa, N.: Face detection using coarse-to-fine support vector classifiers. In: International Conference on Image Processing, pp. 925–928 (2001)
16. Romdhani, S., Torr, P., Schölkopf, B., Blake, A.: Efficient face detection by a cascaded support-vector machine expansion. R. Soc. Lond. Proc. Ser. A **460**(2051), 3283–3297 (2004)
17. Ojala, T., Pietikainen, M., Harwood, D.: A comparative study of texture measures with classification based on featured distributions. Pattern Recognit. **26**(1), 51–59 (1996)
18. Hadid, A., Pietikainen, M., Ahonen, T.: A Discriminative Feature Space for Detecting and Recognizing Faces. In: IEEE Conference on Computer Vision and Pattern Recognition (2004)
19. Pietikainen, M., Abdenour, H., Zhao, G., Ahonen, T.: Computer Vision Using Local Binary Patterns, vol. 40. Springer-Verlag, London (2011)
20. Catanzaro, B., Sundaram, N., Keutzer, K.: Fast support vector machine training and classification on graphics processors. In: International conference on Machine Learning, pp. 104–111 (2002)
21. Zhao, H.-X., Magoules, F.: Parallel Support Vector Machines on Multi-core and Multiprocessor Systems. In: International Conference on Artificial Intelligence and Applications (2010)
22. Anguita, D., Boni, A., Ridella, S.: A digital architecture for support vector machines: theory, algorithm, and FPGA implementation. IEEE Trans. Neural Netw. **14**(5), 993–1009 (2003)
23. Genov, R., Gauwengerghs, G.: Kerneltron: support vector machines in silicon. IEEE Trans. Neural Netw. **14**, 1426–1434 (2003)
24. Mahmoodi, D., Soleimani, A., Khosravi, H., Taghizadeh, M.: FPGA simulation of linear and nonlinear support vector machine. J. Softw. Eng. Appl. **4**, 320–328 (2011)
25. Biasi, I., Boni, A., Zorat, A.: A reconfigurable parallel architecture for SVM classification. In: IEEE International Joint Conference on Neural Networks, pp. 2867–2872 (2005)
26. Groleat, T., Arzel, M., Vaton, S.: Harware Acceleration of SVM-based traffic classification on FPGA. In: International Wireless Communications and Mobile Computing Conference, pp. 443–449 (2012)
27. Ruiz-Llata, M., Yebenes-Calvino, M.: FPGA Implementation of Support Vector Machines for 3D Object Identification. In: International Conference on Artificial Neural Networks (2009)
28. Reyna-Rojas, R., Houzet, D., Dragomirescu, D., Carlier, D., Ouadjaout, S.: Object recognition system-on-chip using the support vector machines. EURASIP J. Adv. Signal Process. **2005**, 993–1004 (2005)
29. Graf, H.P., et al.: A Massively Parallel Digital Learning Processor. In: Annual Conference on Neural Information Processing Systems (NIPS), pp. 529–536 (2008)
30. Boni, A., Pianegiani, F., Petri, D.: Low-power and low-cost implementation of SVMs for smart sensors. IEEE Trans. Instrum. Meas. **56**(1), 39–44 (2007)

31. Anguita, D., Ghio, A., Pischiutta, S., Ridella, S.: A Hardware-friendly Support Vector Machine for Embedded Automotive Applications. In: International Joint Conference on Neural Networks (2007)
32. Meher, P.K., Valls, J., Tso-Bing, J., Sridharan, K., Maharatna, K.: 50 Years of CORDIC: Algorithms, Architectures, and Applications. In: IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 56, no. 9, pp. 1893–1907 (2009)
33. Khan, F., Arnold, M., Pottenger, W.: Finite Precision Analysis of Support Vector Machine Classification in Logarithmic Number Systems. In: Euromicro Symposium on Digital System Design (2004)
34. Khan, F., Arnold, M., Pottenger, W.: Hardware-based support vector machine classification in logarithmic number systems. In: IEEE International Symposium on Circuirs and Systems (2005)
35. Boni, A., Zorat, A.: FPGA Implementation of Support Vector Machines with Pseudo-Logarithmic Number Representation. In: International Joint Conference on Neural Networks (2006)
36. Anguita, D., Ghio, A., Pischiutta, S.: A learning machine for resource-limited adaptive hardware. In: Second NASA/ESA Conference on Adaptive Hardware and Systems (2007)
37. Ghio, A., Pischiutta, S.: A Support Vector Machine based pedestrian recognition system on resource-limited hardware architectures. In: Research in Microelectronics and Electronics Conference (2007)
38. Anguita, D., Pischiutta, S., Ridella, S., Sterpi, D.: Feed-forward support vector machine without multipliers. IEEE Trans. Neural Netw. **17**, 1328 (2006)
39. Carpenter, A.: CUSVM: A CUDA Implementation of Support Vector Machines (2009)
40. Bauer, S., Kohler, S., Doll, K., Brunsmann, U.: FPGA-GPU Architecture for Kernel SVM Pedestrian Detection. In: Computer Vision and Pattern Recognition Workshops (2010)
41. Papadonikolakis, M., Bouganis, C.-S.: Novel cascade FPGA accelerator for support vector machines classification. Trans. Neural Netw. Learn. Syst. **23**(7), 1040–1052 (2012)
42. Maghazeh, A., Bordoloi, U., Eles, P., Peng, Z.: General Purpose Computing on Low-Power Embedded GPUs : Has It Come of Age? In: International Conference on Embedded Computer Systems (SAMOS XIII) (2013)
43. Kyrkou, C., Theocharides, T.: A parallel hardware architecture for real-time object detection with support vector machines. IEEE Trans. Comput. **61**(6), 831–842 (2012)
44. Kryjak, T., Komorkiewicz, M., Gorgon, M.: FPGA implementation of real-time head-shoulder detection using local binary patterns, SVM and foreground object detection. In: International Conference on Design and Architectures for Signal and Image Processing (2012)
45. Microblaze Soft Processor. Xilinx, San Jose, CA. http://www.xilinx.com/tools/microblaze.htm
46. CBCL Face Database #1, MIT Center for Biological and Computation Learning. http://cbcl.mit.edu/software-datasets/FaceData2.html
47. CBCL PEDESTRIAN DATABASE #1, MIT Center for Biological and Computation Learning. http://iris.usc.edu/Vision-Users/OldUsers/bowu/DatasetWebpage/dataset.html
48. Bao Face Database. http://www.facedetection.com/downloads/BaoDataBase.zip
49. CMU and MIT Face Database. http://vasc.ri.cmu.edu/idb/html/face/frontal_images/
50. PETS 2012 Database. In: 14th IEEE International Workshop on Performance Evaluation of Tracking and Surveillance. http://www.pets2012.net/