Indonesian Journal of Electrical Engineering and Computer Science
Vol. 16, No. 3, December 2019, pp. 1286∼1296
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v16i3.pp1286-1296

A streaming multi-class support vector machine classification architecture for embedded systems

Jeevan Sirkunan, J.W. Tang, N. Shaikh-Husin, M. N. Marsono
School of Electrical Engineering, Faculty of Engineering, Universiti Teknologi Malaysia, Johor, Malaysia

Article Info

Article history:

Received Dec 28, 2018
Revised Jun 1, 2019
Accepted Jun 28, 2019

Keywords:

Multi-class SVM
FPGA
embedded system

ABSTRACT

Pedestrian detection, face detection, speech recognition and object detection are some of the applications that have benefited from hardware-accelerated Support Vector Machine (SVM). Computational complexity of SVM classification makes it challenging for designing hardware architecture with real-time performance and low power consumption. On an embedded streaming architecture, testing data are mostly stored on external memory. Data are transferred in streams with the maximum bandwidth limited to the bus bandwidth. The hardware implementation for SVM classification needs to be sufficiently fast to keep up with the data transfer speed. Prior implementation throttles data input to avoid overwhelming the computational unit. This results in a bottleneck in the streaming architecture. In this work, we propose a streaming-architecture multi-class SVM classification for an embedded system that is fully pipelined and able to process data continuously without any need to throttle data stream input. The proposed design is targeted for embedded platform where testing data is transferred in streams from external memories. The architecture is modeled using Verilog and the evaluation is targeted for Altera Cyclone IV field programmable gate array platform. Performance profiling on the proposed architecture is done with regard to the number of features and support vectors. For validation, the proposed architecture is simulated using ModelSim and the results are compared with LibSVM. Based on the simulation result, the proposed architecture is able to produce a throughput of $1/N_f$ classification per clock cycle, where N_f is the number of features.

Corresponding Author:

Jeevan Sirkunan,
School of Electrical Engineering,
Faculty of Engineering, Universiti Teknologi Malaysia,
Johor, Malaysia.
Email: jeevan2@live.utm.my

1. INTRODUCTION

Pervasive computing leads to emergence of applications that mandated computational intelligence and analytics. Entertainment, sensor networks, health care and environmental monitoring are some of the example applications that have embedded pervasive computing into their system [1]. These applications need to process large amounts of data, and this requires massive data parallelism that needs high data bandwidth transfer between the processors and off-chip memory. Such data access pattern renders on-chip caches mostly ineffective [2]. At the same time, some of these applications require a low size, low power processor with real-time operating performance especially for an embedded system [1]. There are several ways in dealing with this demand. One method is to use a reconfigurable hardware such as field programmable gate array (FPGA) that provides high performance computation at minimal cost overhead and power consumption [3].

Support Vector Machine (SVM) is an accurate binary classifier that is based on a solid theoretical background [4-6]. Some applications that have benefited from SVM hardware acceleration are pedestrian detection [7], face detection [8], speech recognition [9] and object detection [10, 11]. Due to SVM computational complexity, designing a hardware architecture with low power consumption and real-time classification performance still remains a challenge [12]. Many existing custom SVM hardware implementations focused on accelerating the decision function that is application specific [8, 13]. These designs cannot be easily reused for other applications as they are optimized for specific applications [8]. Works that targeted on acceleration of SVM with a co-processor unit [2, 14, 15, 16] tend to focus on the kernel due to its compute-intensive task and also its innate nature to be parallelized.

The performance of a single accelerator unit or a co-processor unit is highly dependent on the rate of data transfer. With limited on-chip memory in an embedded system, data are stored in off-chip memory. Data are transferred in streams and the maximum bandwidth depends on the bus width of a particular platform. Kane et al. [12] proposed a fully-pipelined SVM architecture that supports multi-class classification which was tested with a wide range of data sets. However, in their implementation input throttling was required to avoid data input from overwhelming the processing unit. The throughput of their proposed architecture is limited under high load.

In this work, we propose a multi-class fully pipelined SVM classification streaming architecture for embedded system. The architecture is able to produce output at the same rate as the data input without the need for data input throttling. The proposed hardware architecture is based on LibSVM model data structure. Performance evaluation of the proposed hardware implementation is done with regard to the number of support vectors and features. Results show that the proposed design has an initial latency approximately equivalent to the number of support vectors and able to process data continuously without any need to throttle the data transfer. This feature is crucial as training data are transferred in streams from external memory.

2. RESEARCH METHOD

Traditionally, there have been two fundamentally different types of tasks in machine learning: unsupervised and supervised. The goal of unsupervised learning is to find patterns and structure in the data, while supervised learning is to learn a mapping from a labeled data set [18]. SVM falls under the category of the latter. In general, SVM tasks can be divided into two: training and classification. Based on a labeled data set, SVM is trained to obtain its decision function. With the decision function, unknown data can then be classified.

2.1. Training

SVM is inherently a binary classifier; in the training phase, it determines the decision boundary that maximizes the space between two classes [19]. In order to handle data which cannot be separated linearly in low-dimensional feature space, kernel functions are used to project learning data into high-dimensional space so that it can be linearly separated [20]. Equations (1) and (2) show the dual Lagrange problem to obtain the SVM decision function. $K(x_i, x_j)$ is the kernel function, b is the bias parameter and C is the trade-off parameter between maximizing the margin and minimizing the error.

$$\text{Maximize } L = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j K(x_i, x_j)$$
$$\text{Subject to} \begin{cases} 0 \le \alpha \le C \\ \sum_{i=1}^{N} \alpha_i y_i = 0 \end{cases} \tag{1}$$

$$b = y_i - \Big(\sum_{i=1}^{N} \alpha_i x_i y_i\Big) \tag{2}$$

where :

α	Lagrangian multiplier
N	Total amount of training data set
x_i	Training data (feature set)
y_i	Training data (label)

The main aim of the training functions (equations (1) and (2)) are to obtain α_i and b values. If α_i is 0, the corresponding training feature vector $\mathbf{x}_i$ is not a support vector. If α_i is very high, then the corresponding training feature $\mathbf{x}_i$ has high influence over the decision surface of the hyperplane ($\mathbf{x}_i$ becomes a support vector).

2.2. Classification

Equation (3) shows the SVM decision function $d(u)$ for classification. The computation only extends to the number of support vectors N_{sv}, where the support vectors are a subset of the training data. The class of unknown input u is determined by the sign of the function in equation (3).

$$d(u) = \text{sign}\Big(\sum_{i=1}^{N_{sv}} \alpha_i K(x_i, u) + b\Big) \tag{3}$$

2.3. Kernel Function

Kernel functions efficiently map non-linear datasets to a high dimensional linear feature space. Kernel function allows SVM to handle non-linear data. The type of kernel to be employed is entirely dependent on the characteristics of the application data sets, which is beyond the scope of this paper. Some of the main kernel functions that are used in literature are linear (equation (4)), polynomial (equation (5)), Gaussian radial function (RBF) (equation (6)) and sigmoid (equation (7)).

$$K(x_i, u) = x_i \cdot u \tag{4}$$
$$K(x_i, u) = (\alpha(x_i \cdot u) + r)^d \tag{5}$$
$$K(x_i, u) = e^{(-\gamma)x_i - u^2}, \text{where } \gamma > 0 \tag{6}$$
$$K(x_i, u) = \tanh(\alpha(x_i \cdot u) + r) \tag{7}$$

3. RELATED WORKS

Extant works on accelerating SVM on FPGA can be divided into two main groups: training and classification. Accelerating training phase focuses on certain applications where online learning is required. Applications such as adaptive channel equalization [21] and sketch recognizer [22] demand short training time for fast adaptation since the characteristics of data change over time. On the other hand, accelerating classification phase is important when a task requires a fixed classification module with large data to be classified [8, 13].

Sequential Minimal Optimization (SMO) [23] is considered as one of the commonly used algorithms for optimizing SVM training problem. However, the SMO algorithm itself was designed in such a way that it caters for single-threaded computer [16]. Besides SMO, there are also Gilbert's algorithm [24] and Least Squared Support Vector Machine (LS-SVM) [25] for training. Works that are targeted for training either implement the whole system-on-chip or offloaded to a co-processor [2, 14, 15, 16] to accelerate the training process. The task that is targeted for hardware implementation is the kernel as it is a compute-intensive task that benefits from parallelization. SVM training phase produces a model, which is used in the classification phase later. This model then dictates the architecture, logic resource and memory bits utilization of the hardware implementation for SVM classification.

For the classification part, many existing custom hardware implementations focused on accelerating the decision function for a specific task [8, 9, 11, 13, 26]. These designs cannot easily be reused for other purposes as these were targeted for a fixed number of support vectors and classes. Another approach towards SVM classification on FPGA system is cascaded SVM [27]. The cascaded SVM architecture by [27] contains a combination of a high and a low precision module that are implemented depending on the rate of the incoming testing data. The low precision module processes incoming data since it has higher throughput potential. Data which are not able to be classified with certainty is transferred to the high precision module which has lower throughput potential but is able to classify the data more accurately. However, [27] focused was solely on the binary classification problem.

The implementation of SVM classification in hardware requires many multipliers especially the kernel modules. This has led to the use of the CORDIC algorithm in implementing SVM classification accelerator module. Ruiz-Llata et al. [1] proposed an SVM classification with multi-class support using the

CORDIC algorithm to replace multipliers [1]. However, timing performance was not reported and the work was not benchmarked with any CPU or GPU implementation. As the CORDIC method is iterative, this would result in a reduction in performance for more complex data sets with a larger amount of support vectors and classes. CORDIC unrolled loop can be implemented with fixed pipelined stages. However, this would consume additional hardware and would affect accuracy for certain input conditions. Multiplications in FPGA were avoided in earlier implementations since it was resource-hungry. However, newer FPGA families have dedicated hardware multipliers and MAC units [17].

Kane et al. [12] proposed a fully pipelined SVM decision architecture for multi-class with a generic design, which was then tested with a wide variety of datasets. This work performed better than GPU and CPU implementations in terms of throughput with practically a negligible trade-off in accuracy. However, in their proposed work, a stalling mechanism was used to stop input data from overflowing the processing unit. The rate of input data needs to be throttled to match with the maximum rate of the data output of the proposed hardware.

In this work, we propose a streaming architecture for embedded systems that is fully pipelined and is able to operate without the need to throttle the rate of input data. This allows the proposed architecture to achieve maximum allowable throughput depending on the rate of data input. The proposed architecture is targeted for embedded system platform where training data are stored in external memory source and continuously streamed-in through a fixed memory bandwidth. Training data are computed as they arrive, which result in the throughput of the data output to be equivalent to the data input. The architecture has an initial latency approximately equivalent to the number of support vectors.

4. LIBSVM MODEL

The proposed architecture is based on the model generated by LibSVM [28]. During the training phase, based on a set of training data, LibSVM creates an SVM model file. This file contains important parameters for the decision function. Besides that, the classifier model also contains information on types of classification, kernel type, kernel parameters, number of support vectors from each class and a bias value. For each binary training problem, LibSVM generates a set of α that is used in the decision function. There are many multi-class approaches in SVM, such as pairwise, DAG, and one-versus-all. LibSVM uses the pairwise method. Therefore for each multi-class SVM model, there will be $\frac{n!}{2(n-2)!}$ sets of α coefficient for computation, where n is the number of classes.

Figure 1 shows the format for the SVM classifier model with four classes as generated by LibSVM. N_{sv} represents the number of support vectors, N_f is the number of features, and N_c is the number of classes. During the training phase, each support vector is compared with other support vectors to ensure that no redundant SVs are stored in the model. In the coefficient column for SV belonging to another binary classier, the α coefficient is set to zero. Based on this structure, each data input can be multiplied with all SVs and α coefficients without requiring a multiplexer for selection in the SVM hardware module.

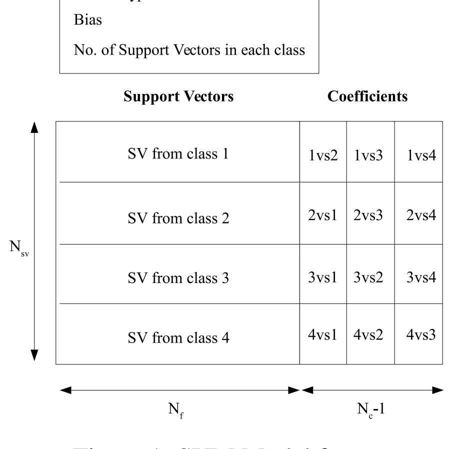


Figure 1. SVM Model format

5. PROPOSED ARCHITECTURE

This section presents the proposed architecture. Figure 2. shows the overview of the architecture. The architecture is divided into four sections: kernel computation, α coefficient weighting, voting summation, and vote sorting. Each of this module is structured in such a way to produce output at every clock cycle. Our implementation design is parameterizable in terms of N_{sv} and N_f in order to handle various types of application. The input of the proposed architecture is an external memory connected to the FPGA. Unlike on-chip memory where data can be accessed simultaneously, for external memory, the bandwidth is limited. For FPGA devices that have higher memory transfer bandwidth with external memory, the proposed design can be expanded with the same structure to support the high bandwidth data input. For our implementation, we fixed the input to be one feature per clock cycle.

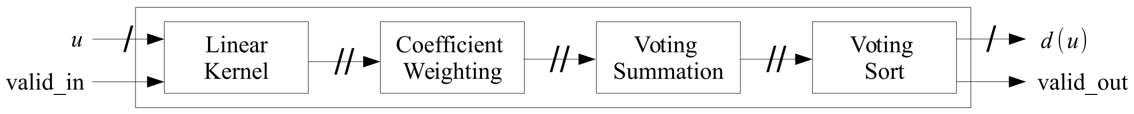


Figure 2. Proposed architecture overview

5.1. Kernel Computation

This module performs kernel computation as the input data is streamed-in continuously every clock cycle, resulting in the throughput of the system to be $1/N_f$. In each clock cycle, a single feature is transferred to the module.

Figure 3 shows the overview of the linear kernel architecture. The number of multipliers needed is based on the number of SVs that are available in the model. The linear kernel implementation is based on LibSVM. This structure is maintained so that it can easily be substituted with a different type of kernel for future implementation. From the SVM model in Figure 1, parallelization of the kernel computation can be done based on either the N_f or the N_{sv}. Parallelizing based on N_f will result in a match with the input data rate. Therefore for proposed design parallelization based on the number of SVs is chosen.

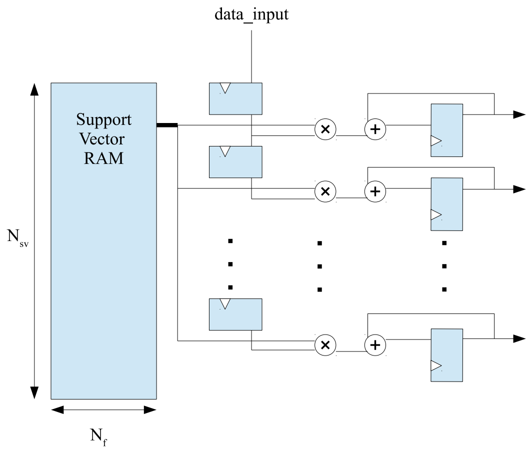
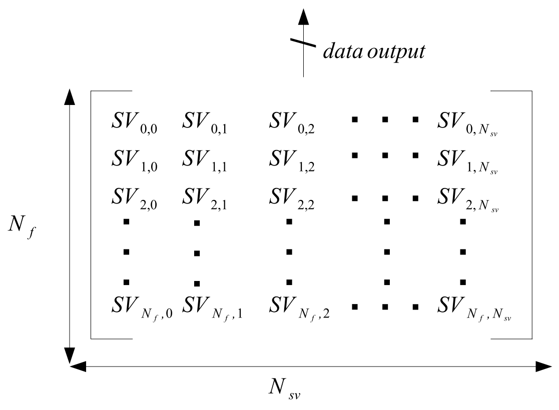


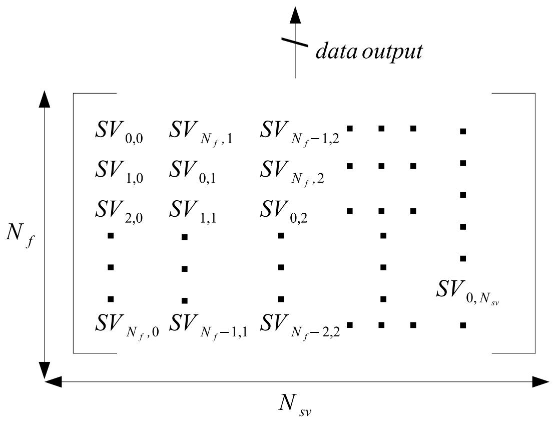
Figure 3. Linear Kernel architecture

In our implementation, each input data is transferred through a shift register in comparison to the work proposed by [26], where input data is broadcasted to each SV at the same time. This is done in order to reduce the fan-out from input data, which spans across the support vector memory output, which would be large for application with many SVs.

Since data traverse through the SV memory with a clock cycle delay, each SV is also accessed with one clock cycle delay. Therefore, in order to avoid multiple individual memory blocks, a counter and a complicated control unit, we concatenate all the support vectors into a single memory block based on the format in Figure 4b. With this scheme, only a single memory block and counter are required. Figure 4a shows the conventional method [28] for storing support vectors in memory. The proposed method (Figure 4b) is based on the conventional method (Figure 4a) where each SV column is shifted one row from left to right.


data output
$SV_{0,0}$ $SV_{0,1}$ $SV_{0,2}$ $SV_{0,N_{sv}}$
$SV_{1,0}$ $SV_{1,1}$ $SV_{1,2}$ $SV_{1,N_{sv}}$
$SV_{2,0}$ $SV_{2,1}$ $SV_{2,2}$ $SV_{2,N_{sv}}$
$SV_{N_f,0}$ $SV_{N_f,1}$ $SV_{N_f,2}$ $SV_{N_f,N_{sv}}$
N_f
N_{sv}

(a) Conventional storage format of support vectors memory [28]


data output
$SV_{0,0}$ $SV_{N_f,1}$ $SV_{N_f-1,2}$
$SV_{1,0}$ $SV_{0,1}$ $SV_{N_f,2}$
$SV_{2,0}$ $SV_{1,1}$ $SV_{0,2}$
$SV_{0,N_{sv}}$
$SV_{N_f,0}$ $SV_{N_f-1,1}$ $SV_{N_f-2,2}$
N_f
N_{sv}

(b) Proposed storage format of support vectors memory

Figure 4. Storage format of support vectors memory

5.2. Coefficient weighting

Based on SVM classification equation (3), resultant kernel computation for each SV will be multiplied with its corresponding coefficient. The resultant is then summed together with its bias. The final result sign is then compared to see if its greater or less than zero. Based on this the class is determined.

Piecewise multi-class approach from [28] compares one class with other classes. During the training phase, one class is assigned to be in the positive class while the other is in the class negative. Generated α coefficients will be based on this notation. The sign on SVM model coefficients for multi-class with four classes is shown in Figure 5. If the α coefficients from 1vs2 are positive, the corresponding 2vs1 would be negative. Based on this, equation (3) is expanded to equation (8). The bias parameter b can be placed on either side of equation (8) depending on its sign. Based on equation (8), the hardware module does not require for signed integer computation.

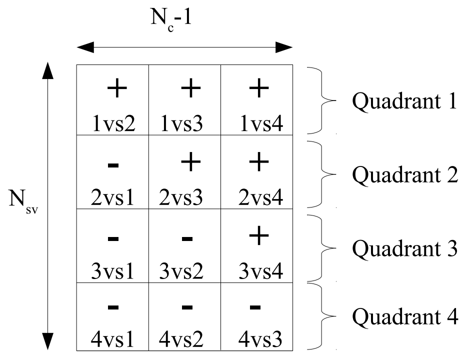


Figure 5. α coefficient sign notation for SVM model with 4 classes

$$\Big(\sum_{i=1}^{N_{sv}+} \alpha_i K(\mathbf{x}_i, \mathbf{x}) \pm b\Big) > \Big(\sum_{i=1}^{N_{sv}-} \alpha_i K(\mathbf{x}_i, \mathbf{x}) \pm b\Big) \tag{8}$$

Figure 6 shows the structure of the α coefficient weighting module. The α values are loaded into the FPGA as constant values. The resultant value from each SV will be summed and stored in each individual

FIFO. The FIFOs are needed for this architecture to store the result from the coefficient weighting module before it is computed in the next module. Testing data are computed from quadrant 1 until 4 (Figure 5). Once the data reaches quadrant 2, comparison for class 1 and 2 can be performed. In quadrant 3, comparison for classes 1 and 2, with 3 can be performed, and finally at quadrant 4, comparison for classes 1,2 and 3 with 4 can be performed. The FIFOs are loaded in depending on the SV that has been completed.

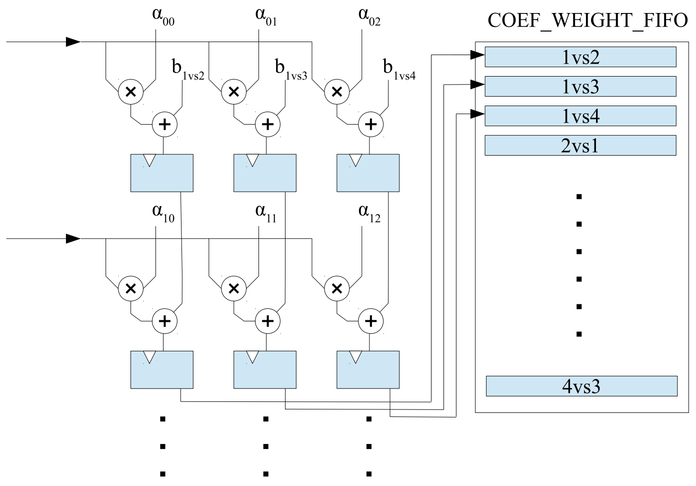


Figure 6. Alpha coefficient weighting architecture

5.3. Voting summation

The outputs of α coefficient weighting module are connected to the comparator to determine the winner for each class comparison (Figure 6). Based on Figure 1, training data inputs are computed with SV from class 1 followed by class 2, class 3 and finally class 4.

The results of each comparator are stored in another set of FIFOs. Figure 7 shows an overview of the voting summation module. The FIFOs are shifted out depending on the SVs that have been completed. The inverter produces an inverted result for the opposing class. Once SVs from all classes are completed, the results are summed up and stored in a register. Each register corresponds to a particular class.

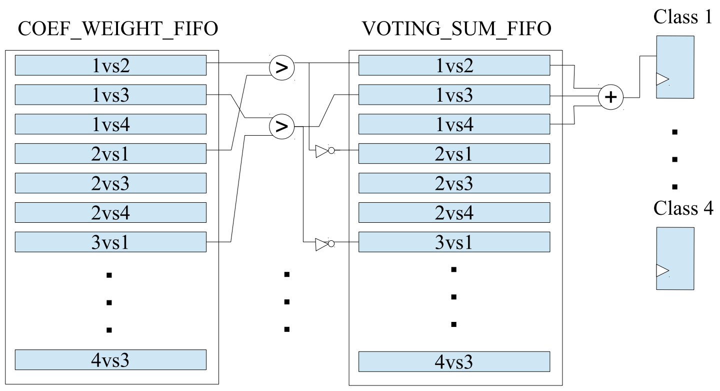


Figure 7. Voting Summation architecture

5.4. Voting sort

Finally, the results of all summations are sorted to determine the class. Figure 8 shows the architecture of the sorting mechanism. Each result register from voting summation is concatenated with the class index. Once sorted, the index will be shown as the labeled class. The architecture is based on bitonic sort [29], but in our proposed architecture it only determines the maximum value within a list. Similar to LibSVM, if two classes show the identical summation result, the class with the smaller index number will be the winner.

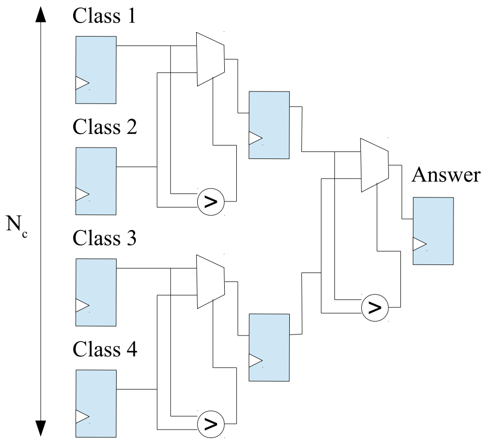


Figure 8. Voting sort architecture

6. RESULT AND DISCUSSION

Depending on the targeted application, the number of SVs, features and class vary. Based on the implementation of [12] with linear kernel for DNA dataset [30], the SVM model has three classes, 180 features and 402 SVs while for Vowel [31] dataset the SVM model has 11 classes, 10 features, and 268 SVs. SVM configurations for certain applications differ based on these parameters. Our analysis has been carried in two scenarios. For Experiment 1, the number of support vectors is fixed to 20 and the size of input data is 16-bit integer. The proposed architecture has been evaluated with a different number of features and SVs. The number of features is adjusted from 10 onwards with increments of 10 until 100 features. For Experiment 2, the number of features is fixed to 32 and the size of input data is also 16-bit integer. The numbers of support vectors are adjusted from 10 onwards with increments of 10 until 100 support vectors. The number of classes is kept constant to four classes. For both experiments, the logic resource utilization, internal memory bits usage, maximum operating frequency (in MHz) and the number of embedded multipliers are analysed. Performance profiling experiment on the proposed architecture was done using Quartus II v13.0 software and the target FPGA device is Cyclone IV EP4CE115F29C7. For validation, the proposed architecture was simulated using ModelSim v10 to obtain cycle accurate results and the results are compared with LibSVM.

Tables 1 and 2 show the results for Experiments 1 and 2, respectively. The number of SVs has a greater impact on logic utilization. As the number of SVs increases, the number of multipliers increases, whereas if the number features increases, only the number of adders will increases. The number of internal memory bits utilization also increases with the number of SVs. This is mainly due to SV memory and the FIFOs that are needed to store the temporary results. The number of SVs does not have a significant impact on the maximum operating frequency (Table 2). The minor differences are mainly due to placement and routing. On the other hand, as the number of features increases, the maximum operating frequency decreases is due to the increase in the size of adders and multipliers for the architecture.

Table 1. Performance evaluation of the proposed architecture towards different number of features.

No. of features	Logic Elements	Memory Bits	Multipliers	Max Frequency (MHz)
10	6408	27776	320	90.46
20	7706	36736	376	85
30	9056	40576	432	79.13
40	10322	54656	488	76.34
50	12429	58496	532	71.78
60	14027	62336	532	68.48
70	16688	86656	532	66.54
80	19223	90496	532	63.4
90	21308	94336	532	60.51
100	23777	98176	532	58.2

Table 2. Performance evaluation of the proposed architecture towards different number of support vectors.

No. of SVs	Logic Elements	Memory Bits	Multipliers	Max Frequency (MHz)
10	4810	20672	244	79.94
20	9300	41344	488	81.31
30	15836	46464	532	80.75
40	24276	82688	532	76.38
50	33080	87808	532	77.22
60	42538	155136	532	74.77
70	52110	160256	532	76.16
80	59679	165376	532	75.41
90	69094	170496	532	71.12
100	80540	175616	532	75.48

The performance evaluations are targeted for a generic SVM classification hardware implementation. A 16-bit data input was assumed to be a 16-bit integer, therefore the adders, multipliers, and registers that follow after the linear kernel module (Figure 3) increase in width in order to maintain a lossless computation. For real application implementation depending on the tolerance of the application towards a lossy result, the proposed architecture can be optimized to have smaller numerical adders and multipliers, which will lower the logic utilization and increase the maximum operating frequency.

For this implementation, the focus was only on the linear kernel. The proposed design can be expanded to different types of the kernel with some minor changes in the kernel computation module. For RBF kernel implementation (equation (6)), exponential computation is required. For a pipelined architecture, a lookup table is more suitable in comparison to CORDIC since it can produce results with a predictable time frame. Based on our architecture, since the input data traversed through the shift register, a shared lookup table for RBF kernel implementation can be used without compromising the throughput of the system. For application where the number of SVs is less than the number of features, the proposed architecture would be able to share a single exponential lookup table. However, if the number of SVs is larger than the number of features, the number of required exponential lookup tables will be equivalent to N_{sv}/N_f. Since N_{sv} determines the number of clock cycles needed for all support vectors to share a single lookup table, whereas N_f represents the number of clock cycles needed for each SV to be computed.

In our performance analysis, the focus was only on the linear kernel. The proposed design can be expanded to different types of kernels with some minor changes in the kernel computation module. For RBF kernel implementation (Eq.) exponential computation is required. For a pipelined architecture lookup table is more suitable in comparison to CORDIC since its able to produce results with a predictable time frame. Based on our architecture, since the input data traversed through the shift register, a shared lookup table for RBF kernel implementation can be used without compromising the throughput of the system. For application where the number of SVs is less then the number of features, the proposed architecture would be able to share a single exponential lookup table. However, if the number of SVs is larger then the number of features the number of exponential lookup table needed will be equivalent to N_{sv}/N_f. N_{sv} determines the number of clock cycle needed for all SVs to share a single lookup table whereas N_f represents the the number of clock cycle needed for each SV to be computed before the next input data starts to be computed.

7. CONCLUSION

The proposed streaming architecture for SVM classification is able to produce output at the same rate as data input. The throughput of the implementation is equivalent to $1/N_f$. Based on the performance profiling experiments, the proposed architecture resource utilization is dependent on the number of SVs, while the e resource utilization is dependent on the number number of features determines the maximum operating frequency. For future work, the proposed architecture needs to be tested with real-world data sets. From this, the trade-off of accuracy between the software and hardware implementations can be observed. Besides that, explorations on different streaming hardware architecture for different multi-class SVM method need to be done. Other multi-class approaches can be applied to the same problem, which can result in different hardware utilization and accuracy.

ACKNOWLEDGEMENT

This work is supported in part by the Collaborative Research in Engineering, Science & Technology (CREST) grant P17C114 (vote no. 4B176) and Universiti Teknologi Malaysia Matching grant (vote no. 00M75).

REFERENCES

[1] Ruiz-Llata M, Guarnizo G, Yébenes-Calvino M. FPGA implementation of a support vector machine for classification and regression. In Proceedings of the 2010 IEEE International Joint Conference on Neural Networks (IJCNN), 2010 (pp. 1-5).

[2] Cadambi S, Durdanovic I, Jakkula V, Sankaradass M, Cosatto E, Chakradhar S, Graf HP. A massively parallel FPGA-based coprocessor for support vector machines. In Proceedings of the 2009 17th IEEE Symposium on Field Programmable Custom Computing Machines, 2009 (pp. 115-122).

[3] Véstias MP. High-performance reconfigurable computing granularity. Encyclopedia of Information Science and Technology, Third Edition. IGI Global. 2015:3558-3567.

[4] Kotsiantis SB, Zaharakis I, Pintelas P. Supervised machine learning: A review of classification techniques. Emerging Artificial Intelligence Applications in Computer Engineering. 2007:160:3-24.

[5] Vapnik V. The nature of statistical learning theory. Springer science & business media, 2013.

[6] Mahmoodi D, Soleimani A, Khosravi H, Taghizadeh M. FPGA simulation of linear and nonlinear support vector machine. Journal of Software Engineering and Applications. 2011, 4(05):320.

[7] Hahnle M, Saxen F, Hisung M, Brunsmann U, Doll K. FPGA-based real-time pedestrian detection on high-resolution images. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2013 (pp. 629-635).

[8] Kyrkou C, Bouganis CS, Theocharides T, Polycarpou MM. Embedded hardware-efficient real-time classification with cascade support vector machines. IEEE Transactions on Neural Networks and Learning Systems. 2015, 27(1):99-112.

[9] Cutajar M, Gatt E, Grech I, Casha O, Micallef J. Hardware-based support vector machine for phoneme classification. In Proceedings of the 2013 IEEE International Conference on Smart Technologies (EUROCON), 2013 (pp. 1701-1708).

[10] Mizuno K, Terachi Y, Takagi K, Izumi S, Kawaguchi H, Yoshimoto M. An FPGA implementation of a HOG-based object detection processor. IPSJ Transactions on System LSI Design Methodology. 2013, 6:42-51.

[11] Qasaimeh M, Sagahyroon A, Shanableh T. FPGA-based parallel hardware architecture for real-time image classification. IEEE Transactions on Computational Imaging. 2015, 1(1):56-70.

[12] Kane J, Hernandez R, Yang Q. A reconfigurable multiclass support vector machine architecture for real-time embedded systems classification. In Proceedings of the 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2015 (pp. 244-251).

[13] Kryjak T, Komorkiewicz M, Gorgon M. FPGA implementation of real-time head-shoulder detection using local binary patterns, SVM and foreground object detection. In Proceedings of the 2012 IEEE Conference on Design and Architectures for Signal and Image Processing, 2012 (pp. 1-8).

[14] Bustio-Martínez L, Cumplido R, Hernández-Palancar J, Feregrino-Uribe C. On the Design of a Hardware-Software Architecture for Acceleration of SVM's Training Phase. In Proceedings of the Mexican Conference on Pattern Recognition, 2010 (pp. 281-290).

[15] Wang JF, Peng JS, Wang JC, Lin PC, Kuan TW. Hardware/software co-design for fast-trainable speaker identification system based on SMO. In Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics, 2011 (pp. 1621-1625).

[16] Venkateshan S, Patel A, Varghese K. Hybrid working set algorithm for SVM learning with a kernel coprocessor on FPGA. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2015, 23(10):2221-32.

[17] Škoda P, Rogina BM, Sruk V. FPGA implementations of data mining algorithms. In Proceedings of the 35th International Convention of Information Communication Technology (MIPRO), 2012 (pp. 362-367).

[18] Chapelle O, Scholkopf B, Zien A. Semi-supervised learning. IEEE Transactions on Neural Networks. 2009, 20(3):542.

[19] Vapnik V. Statistical Learning Theory. John Wiley&Sons. Inc. 1998:156-60.

[20] Song X, Wang H, Wang L. FPGA implementation of a support vector machine based classification system and its potential application in smart grid. In Proceedings of the 11th International Conference on Information Technology: New Generations, 2014, (pp. 397-402).

[21] Sebald DJ, Bucklew JA. Support vector machine techniques for nonlinear equalization. IEEE Transactions on Signal Processing. 2000, 48(11):3217-3226.

[22] Sugano Y, Matsushita Y, Sato Y, Koike H. An incremental learning method for unconstrained gaze estimation. In Proceedings of the European Conference on Computer Vision, 2008 (pp. 656-667).

[23] Keerthi SS, Shevade SK, Bhattacharyya C, Murthy KR. Improvements to Platt's SMO algorithm for SVM classifier design. Neural Computation. 2001, 13(3):637-49.

[24] Rabieah MB, Bouganis CS. FPGA based nonlinear support vector machine training using an ensemble learning. In Proceedings of the 25th IEEE International Conference on Field Programmable Logic and Applications (FPL), 2015 (pp. 1-4).

[25] Wang S, Peng Y, Zhao G, Peng X. Accelerating on-line training of LS-SVM with run-time reconfiguration. In Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT) 2011, (pp. 1-6).

[26] Koide T, Hoang AT, Okamoto T, Shigemi S, Mishima T, Tamaki T, Raytchev B, Kaneda K, Kominami Y, Miyaki R, Matsuo T. FPGA implementation of type identifier for colorectal endoscopie images with NBI magnification. In Proceedings of the IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), 2014 (pp. 651-654).

[27] Papadonikolakis M, Bouganis CS. Novel cascade FPGA accelerator for support vector machines classification. IEEE Transactions on Neural Networks and Learning Systems. 2012, 23(7):1040-1052.

[28] Chang CC, Lin CJ. LIBSVM: A library for support vector machines. ACM transactions on Intelligent Systems and Technology (TIST). 2011, 2(3):27.

[29] Batcher KE. On Bitonic Sorting Networks. In Proceedings of the International Conference on Parallel Processing (ICPP), 1990 (pp. 376-379).

[30] Michie D, Spiegelhalter DJ, Taylor CC. Machine learning. Neural and Statistical Classification. 1994:13.

[31] Frank A. UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. http://archive.ics.uci.edu/ml. 2010.