electronics

MDPI

Article

Algorithmic-Level Approximate Tensorial SVM Using High-Level Synthesis on FPGA

Hamoud Younes [1,2], Ali Ibrahim [1,2,*], Mostafa Rizk [2] and Maurizio Valle [1]

[1] Department of Electrical, Electronic and Telecommunications Engineering and Naval Architecture, University of Genova, 16145 Genova, Italy; Hamoud.Younes@edu.unige.it (H.Y.); Maurizio.Valle@unige.it (M.V.)

[2] Department of Computer and Communication Engineering, Lebanese International University, Beirut 1105, Lebanon; Mostafa.rizk@liu.edu.lb

* Correspondence: Ali.Ibrahim@edu.unige.it; Tel.: +39-3279-364-917

Abstract: Approximate Computing Techniques (ACT) are promising solutions towards the achievement of reduced energy, time latency and hardware size for embedded implementations of machine learning algorithms. In this paper, we present the first FPGA implementation of an approximate tensorial Support Vector Machine (SVM) classifier with algorithmic level ACTs using High-Level Synthesis (HLS). A touch modality classification framework was adopted to validate the effectiveness of the proposed implementation. When compared to exact implementation presented in the state-of-the-art, the proposed implementation achieves a reduction in power consumption by up to 49% with a speedup of 3.2×. Moreover, the hardware resources are reduced by 40% while consuming 82% less energy in classifying an input touch with an accuracy loss less than 5%.

Keywords: approximate computing; embedded machine learning; tensorial kerne; high-level synthesis; tactile sensing

check for updates

Citation: Younes, H.; Ibrahim, A.; Rizk, M.; Valle, M. Algorithmic-Level Approximate Tensorial SVM Using High-Level Synthesis on FPGA. *Electronics* **2021**, *10*, 205. https://doi.org/doi:10.3390/electronics10020205

Received: 31 December 2020
Accepted: 15 January 2021
Published: 17 January 2021

1. Introduction

Machine Learning (ML) algorithms are efficient solutions for various tasks including speech recognition, tactile data classification and image processing. Consequently, embedding machine learning is increasingly used in several application domains such as prosthesis, Internet of Things (IoT), robotics, smart appliances and wearable devices. Support Vector Machine (SVM) is one of the most used supervised algorithms as it exploits complex relationships among data samples by using "Kernels" to create an optimal hyperplane that separates different classes [1]. Despite having a high classification accuracy, SVM is characterized by its computational complexity. Thus, hardware implementations dedicated to SVM impose additional overhead in terms of power consumption and execution time [2]. This overhead adds a design challenge when targeting real-time embedded systems. Several hardware implementations have been presented using different computing platforms that fulfill the requirements in terms of limited hardware resources, low power consumption and low latency. These platforms include Advanced RISC Machines (ARM), Application-Specific Integrated Circuits (ASIC) and Field-Programmable Gate Array (FPGA). The authors of [3] showed that the implementation of the tensorial SVM (TSVM) algorithm on an ARM Cortex M4 microcontroller (STM32F405) operating at 165 MHz classifies an input touch in 7 s. The latter is higher than the classification time obtained using the FPGA device presented in [4], which is about 400 ms. As for ASIC, the implementation of machine learning inference is characterized by several challenges such as: reconfigurability and design cycle complexity. On the other hand, FPGAs are suitable for prototyping where their features of programmability make them more cost-effective than ASICs. Thus, FPGAs are suitable platforms for implementing such algorithms. In addition, FPGAs are desirable for ASIC prototyping. In this perspective, FPGA has been proposed as a hardware platform for implementing SVM due to its powerful and parallel processing as a re-configurable device with an efficient utilization of hardware resources [2].

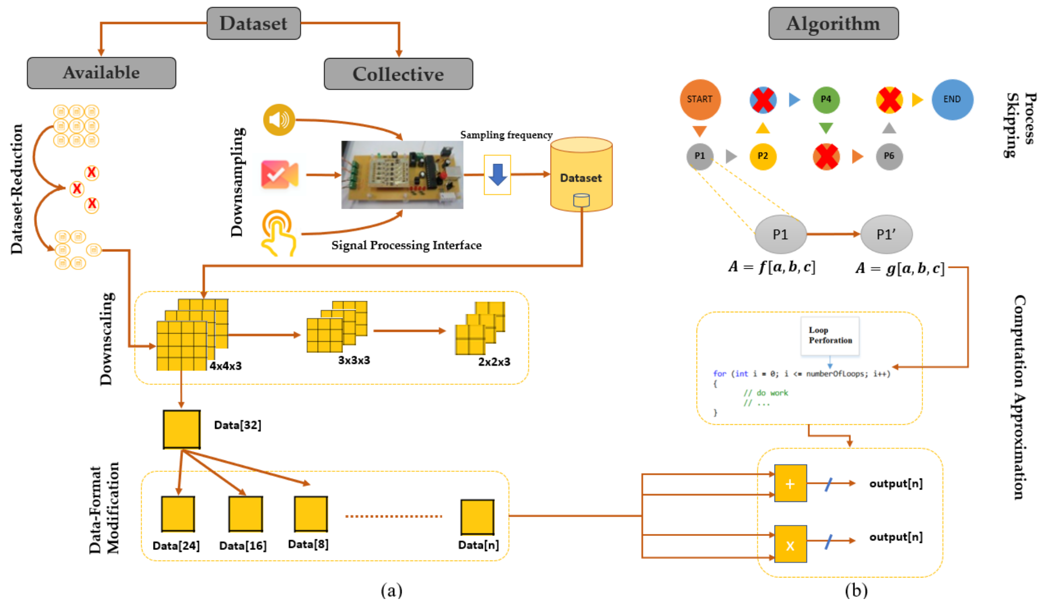
The authors of [5,6] presented a hardware implementation of the linear SVM model that classifies biomedical data targeting FPGA devices. The implementations proposed in [5,6] have achieved a speedup of 85× and 6×, respectively, when compared to similar implementations targeting General-Purpose Processor (GPP). Mandal et al. [7] used a multiplier less-kernel architecture to implement a polynomial based SVM. The architecture leads to a power reduction of 3.5% compared to the use of vector-based kernel. A 2D pipelined streaming architecture with a Radial Basis Function (RBF) kernel implementation is proposed in [8], achieving a 2× speed improvement. In [4], the authors presented an FPGA implementation of SVM based on the tensorial kernel approach, which is proposed in [9]. The advantage of such implementation is that the tensorial representation of data preserves the implicit structure of the original data. Sidiropoulos et al. [10] conducted a survey that demonstrates the importance of using tensorial data in signal processing and machine learning. The survey covers several algorithms and applications, including optimization and statistical performance algorithms, collaborative filtering, classification and multilinear subspace learning. In addition, tensorial representation is recommended for applications such as image and tactile processing [9,11].

Approximate computing has emerged as a promising solution to obtain considerable resource utilization, energy and time savings at the expense of acceptable accuracy loss [12]. There exist notable hardware implementations of SVM in the literature using ACTs on both the algorithmic and circuit levels. Van Leussen et al. [13] presented a hardware architecture with re-configurable kernels and overflow resilient limiter. The proposed architecture achieved a saving of 15% in the consumed energy and 14% in the implementation area for the epileptic seizure detection application compared to a fully-exact implementation. The traditional Ripple-Carry Adder (RCA) has been replaced by an approximate accumulator ,which achieves up to 70% power reduction for the kernel computations in SVM for hyper-spectral image classification problem [14]. The authors of [15] designed an approximate adder and fixed-width multiplier with a low-cost compensation. Adopting the devised adder and multiplier in SVM classifier leads to reduce the power-delay product (PDP), area and critical path delay by 32.4%, 18.7% and 16%, respectively. All the mentioned architectures imply an accuracy loss less than 8% for the target applications. However, to the best of our knowledge, there is currently no implementations of approximate tensorial SVM classifier presented in the literature.

The work presented in this paper aims to reduce the hardware complexity of the tensorial SVM algorithm using algorithmic level ACTs. A touch modality binary classification problem was adopted to validate the proposed implementation. The exact tensorial SVM classifier in [9] as taken as a reference, and then the impact of ACTs on reducing power consumption, hardware resources and time latency was analyzed. High-Level Synthesis (HLS) with Vivado 2018.3 and Virtex-7 FPGA was used for the implementation. HLS design is adopted since it offers: (1) faster development process compared to RTL design; (2) built-in optimization directives; and (3) easier design manipulation through high-level programming languages (e.g., C/C++). The rest of the paper is organized as follows. Section 2 presents the algorithmic level approximate computing techniques and how they are applied on machine learning methods. Section 3 provides a general overview of the tensorial approach. Section 4 details the proposed approximate tensorial SVM architecture and implementation. Section 5 presents an assessment of the hardware implementation results targeting FPGA using HLS. Finally, Section 6 concludes the paper and illustrates the future work.

2. Algorithmic Level Approximate Computing

Approximate computing techniques can be applied at algorithmic, architecture and circuit levels [16]. Algorithmic level techniques are divided into two categories: data-oriented and process-oriented. The authors of [17] presented an approach on how to apply these techniques on machine learning algorithms, as shown in Figure 1.



```
for (int i = 0; i <= numberOfLoops; i++)
{
        // do work
        // ...
}
```

Figure 1. Algorithmic level Approximate Computing Techniques: (**a**) data-oriented; and (**b**) processing-oriented. Adopted from [17].

The data-oriented category involves modifying the data properties (size and bit-width) to minimize the work-load on the circuit level. This category includes:

- Dataset Reduction (DsR) decreases the amount of the processed data by eliminating samples randomly or using a subsampling method as the one proposed in [9]. Furthermore, DsR can be applied through downsampling and downscaling. The former adjusts the sampling frequency of the electronic interface used to collect raw data samples from sensors in the time domain, while the latter reduces the dimension of the collected data themselves (e.g., reducing the tensor size from $4 \times 4 \times 3$ to $3 \times 3 \times 3$), as shown in Figure 1.
- Data Format Modification (DFM) reduces the bit-width of the data and its corresponding arithmetic operations. This can be done by replacing floating-point representation with a fixed-point one. For instance, a 24-bit fixed-point representation data is adopted in [18] instead of floating-point to represent tactile data with a negligible precision loss.

The process oriented category targets the algorithm itself by reducing the number of tasks or replacing some of them with a less-complex counterpart. This category includes [19]:

- Computation Skipping (CS) skips a certain number of tasks in an algorithm. If these tasks are loop iterations, then it is referred to as Loop Perforation (LP). For example, in some machine learning applications, a pre-processing task such as data normalization may be skipped without affecting the quality of service of the target application.
- Computation Approximation (CA) proposes an equivalent version of a computationally complex function. The two versions should be mathematically equivalent with an acceptable output error margin. For example, a division function could be replaced by a reciprocal multiplication [19].

3. Tensorial SVM Algorithm

Gastaldo et al. [9] presented a tensorial kernel approach for touch modalities classification. The approach involves four main steps: (i) Unfolding includes the transformation of a third-order tensor $T(I_1 \times I_2 \times I_3)$ into three matrices $M(I_1 \times I_2 I_3)$, $N(I_2 \times I_1 I_3)$ and $P(I_3 \times I_1 I_2)$. (ii) Singular Value Decomposition (SVD) is used to find the eigenvectors V for each of the three unfolded matrices as:

$$A = USV^T \tag{1}$$

where U and V^T contain the left and right singular vectors, respectively, and S is the diagonal matrix storing the singular values σ_i of A. SVD computations are achieved by using the one sided Jacobi algorithm, which is considered as one of the fastest methods to converge compared to other algorithms [20]. (iii) Kernel computation extends the Gaussian kernels to tensorial patterns. The kernel function can be expressed as:

$$K(x,y) = \prod_{1}^{z} k^z(x,y) \tag{2}$$

where k^z is the kernel factor defined as:

$$k(x,y) = \exp(\frac{-1}{2\sigma^2}(I_n - trace(Z^T Z))) \tag{3}$$

where $Z = V_x^T V_y$, V_x represents the singular vectors of the unfolded matrix, V_y represents the singular vectors of the unfolded matrix obtained from the training phase and trace represents the sum of diagonal elements. (iv) SVM classification applies the classification function expressed in the following equation:

$$y = f_{SVM}(x) = \sum_{i}^{N_p} \beta_i K(x,y) + b \tag{4}$$

where y represents the predicted label of an input x and β_i represents the coefficients obtained during training with a bias b.

4. Approximate Tensorial SVM

The proposed architecture of Approximate SVM is presented in Figure 2. The architecture is an extension to the exact architecture presented in [21]. For the rest of the paper, the latter is referred to as Exact SVM. It involves two stages: offline learning and online inference.

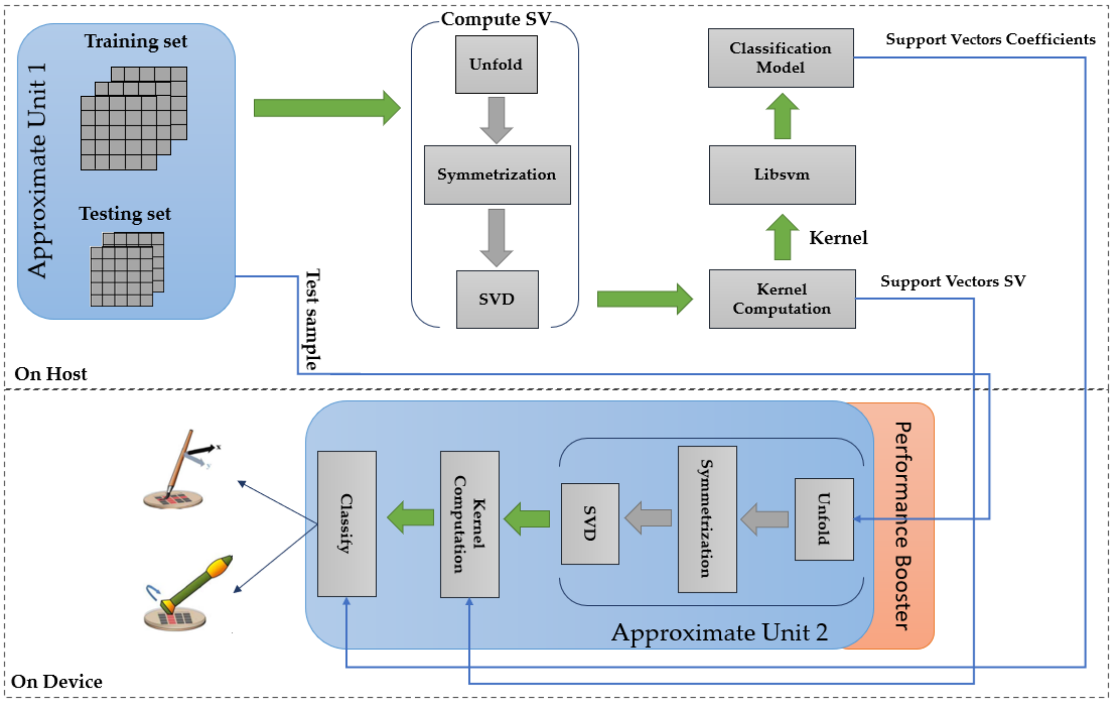


Figure 2. Sketch of the proposed Approximate Tensorial SVM.

4.1. Touch Modality Framework

The touch modality classification problem "Brushing a brush vs. Rolling a washer" referred to as Problem A in [9] was adopted to verify the proposed architecture. This problem is based on a dataset that includes readings from 70 participants. The tactile dataset was acquired by conducting experiments to sense the human touch pressure levels on a 4×4 tactile sensor array for a duration of 10 s with a sampling rate of 3000 samples per second. Thus, each modality can be described by a tensor $\phi(4 \times 4 \times\ 30{,}000)$. The final dataset was split into training and testing sets with $N_t = 180$ and $N_c = 80$. The proposed algorithmic-level approximate computing method discussed in Section 2 was applied on the exact tensorial SVM for the mentioned classification problem. Table 1 shows the effect of using this method on the classification accuracy.

Table 1. Effect of approximate computing techniques on classification accuracy.

Approximate Computing Technique	Accuracy (%)
None (Exact SVM)	90.47
10% Data Set Reduction	90.47
20% Data Set Reduction	80.95
30% Data Set Reduction	80.95
Loop perforation with sf = 2	90.47
Loop perforation with sf = 3	85.71
Loop perforation with sf = 4	80.95
DFM (24-bit)	85.71
DFM (16-bit)	75

The dataset reduction was applied by randomly removing samples from the original dataset. To ensure credible assessment, if a sample is removed during the 10% reduction, it is automatically removed for the 20% and 30% reductions. Loop perforation was applied on the loops of SVD computation block with a skipping factor sf (i.e., how many loops are perforated). As for data format modification, 24- and 16-bit fixed-point representations were applied for all the SVM operations with $< 8, 16 >$ and $< 6, 10 >$ precision, respectively, using the C libraries used in [18]. Based on the obtained results, the training and inference of the exact tensorial SVM were incorporated with the ACTs that resulted in an acceptable trade-off between accuracy and complexity. Combining several ACTs is also known as "cross-layer approximate computing" [12].

4.2. Offline Learning

The SVM training was conducted offline on a PC with Intel i7 CPU. The training process starts by activating the AU1 (Approximate Unit 1) (see Figure 2). AU1 applies dataset-reduction and downscaling techniques on the dataset by performing the following steps:

- The dataset size is reduced by eliminating data that corresponds to five participants with noisy readings. Figure 3a shows an example of such reading where the voltage is almost constant along the measured time. Therefore, the machine learning model will not learn new information from such sample. Hence, it is removed.
- During data collection of the tactile dataset, no precise instructions were given to the participants regarding the amount of pressure to be applied on the sensor [9]. Thus, some touch samples with silent readings where observed such as the one presented in Figure 3b. Such samples could be pre-processed to extract meaningful information in certain time frame. Each sample is truncated from 10 to 3.3 s by omitting readings outside the interval [3.7, 7] s. This results in a new tensor $\phi(4 \times 4 \times 10{,}500)$.
- To reduce the computational complexity of the tensor-based learning algorithms, the tensor size could be reduced without the loss of information originality using

subsampling. The latter is applied by truncating each sample into a new tensor $\phi(4 \times 4 \times 40)$ with 40 random time readings.

Then, the resulting tensor is unfolded into three matrices $M(4 \times 160)$, $N(4 \times 160)$ and $P(40 \times 16)$ that have to be symmetrized before applying SVD. The resulting support vectors along with the Gaussian parameter $\sigma = 1$ are fed to the kernel computation block (see Figure 2). The block outputs the kernel matrices for (+1 vs. −1), (+1 vs. +1), (−1 vs. +1) and (−1 vs. −1) binary classification problems where each row being labeled with the corresponding class label. This step is essential since LIBSVM [22] does not support tensorial kernels by default but can receive precomputed kernels. The LIBSVM library is used to obtain a classification model based on the precomputed kernel. The model contains the coefficients β_i and the bias b.

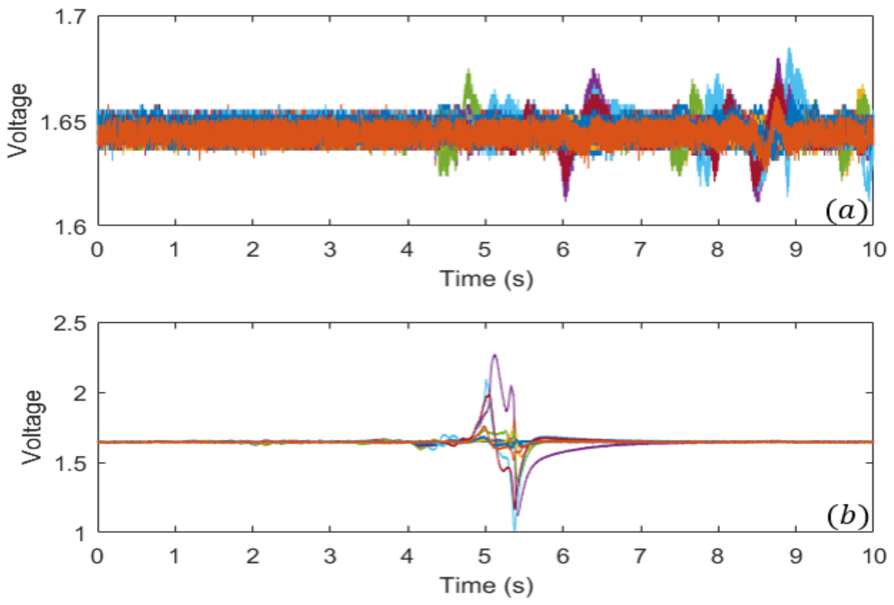


Figure 3. Touch Modalities: (**a**) touch with noisy readings; and (**b**) touch with silent intervals.

4.3. Online Inference

The SVM inference of the proposed architecture was implemented on FPGA through the steps shown in Figure 4. The architecture was coded in C++ using Vivado HLS. Then, the architecture was optimized using HLS directives and synthesized to ensure that it fits in the target FPGA device. Then, a C/RTL simulation was performed to ensure a coherent output from the architecture coded in C++ and the RTL design provided by Vivado HLS. Afterward, the architecture was exported as an RTL IP block targeting a Virtex-7 XC7VX980T FPGA operating at a clock frequency of 120 MHz. The IP block was imported into Vivado; then, a behavioral/combinational simulation was performed to verify the integrity of the exported IP. Then, place and route was performed to implement the architecture on the FPGA device. Finally, a detailed report about the utilized hardware resources, the number of clock cycles and the power consumption was obtained once the implementation was completed.

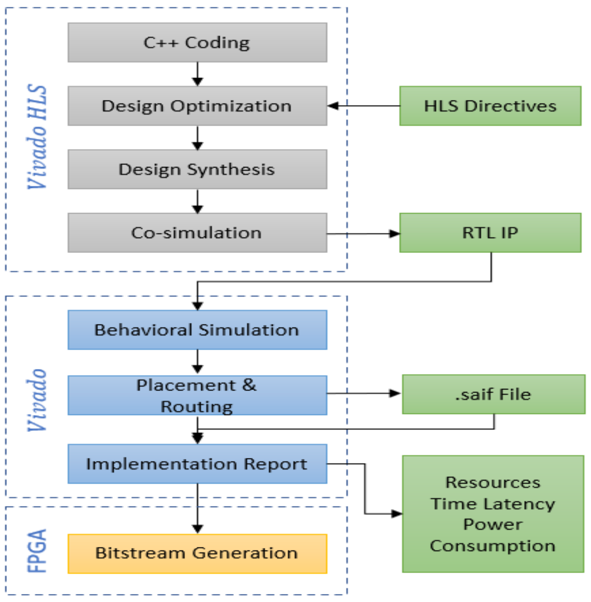


Figure 4. FPGA implementation process.

The inference starts by fetching a sample tensor from the testing set (that was already approximated using AU1). The selected tensor undergoes the unfolding, symmetrization and SVD processes. The obtained support vectors along with the support vectors from the training phase are provided to the kernel computation block. During online inference, Approximate Unit 2 is active. It operates by applying:

- Loop perforation (LP) technique to the SVD block with a skipping factor sf. The support vectors are obtained using the one side Jacobi Algorithm. The latter is an iterative algorithm, thus it is perforated with $sf = 2$. This technique accelerates the SVD computations but a large sf could not be applied to ensure the algorithm's convergence.
- Computation Approximation (CA) to the computation of Z in Equation (3). The obtained singular vector matrices from the SVD block are $V1(160 \times 160)$, $V2(160 \times 160)$ and $V3(16 \times 16)$. These matrices are truncated to $V1'(160 \times 4)$, $V2'(160 \times 4)$ and $V3'(16 \times 2)$. Such truncation reduces the complexity of the matrix multiplication in Equation (3) with an acceptable error margin. This technique was also applied in the offline training phase so that the equation $Z = V_x^T V_y$ has correct dimensions.
- Data Format Modification (DFM) to all the variables and arithmetic operations in different SVM blocks. HLS offers a library called "apfixed", which allows the declaration of variables with fixed-point precision. This declaration is limited by an upper bound [23]. Specifically, the mathematical functions are Square Root ($sqrt$), which is used in SVD calculations, and Exponential (exp), which is used in kernel computations. These functions are supported only for bit-widths $w \leq 32$ and $w \leq 16$, respectively. This limitation was resolved by a variable precision architecture. Hence, all the inference blocks are implemented with 24- bit fixed-point representation with a $< 12, 12 >$ precision except the kernel computation block.

Finally, the output of the kernel computation (i.e., a kernel) is used by the classification block to predict a class for the tested tensor according to Equation (4).

4.4. Performance Booster

The performance of the proposed architecture was enhanced to achieve the lowest possible time latency for applications with timing constraints [24] while increasing the throughput. These requirements are usually accompanied by an increase in hardware resources, but the use of algorithmic level approximate computing techniques, specifically "dataset reduction" and "data format modification", would compensate such increase.

These requirements are facilitated by the use of Vivado HLS optimization directives [23]. The used directives are:

- Array Partition: tThis directive partitions a large BRAM occupied by a multidimensional array into smaller separate memories. The array partitioning can be complete, cyclic or block. The latter was applied on the tensor $\phi(4 \times 4 \times 40)$ with block size = 16, as shown in Figure 5. This results in an RTL IP block with smaller memories while improving the throughput of the Unfolding process.
- Dataflow: This directive allows functions to overlap in their operations, enhancing the overall throughput and latency of the design. The functions unfold and symmetrization are executed in a task-level pipelining using this directive, as shown in Figure 6.
- Pipelining: This directive allows the parallel execution of loop iterations, hence reducing the time latency. The computation of Z in Equation (3) is executed in parallel, as shown in Figure 7.

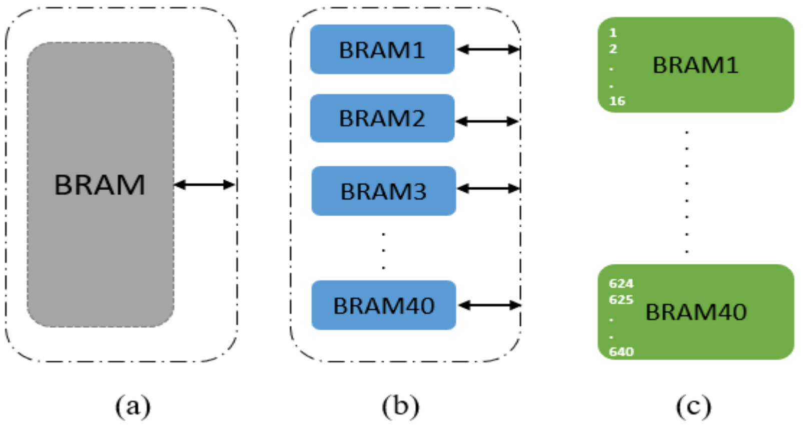


Figure 5. Array partitioning: (**a**) without partitioning; (**b**) block partitioning; and (**c**) block with size=16.

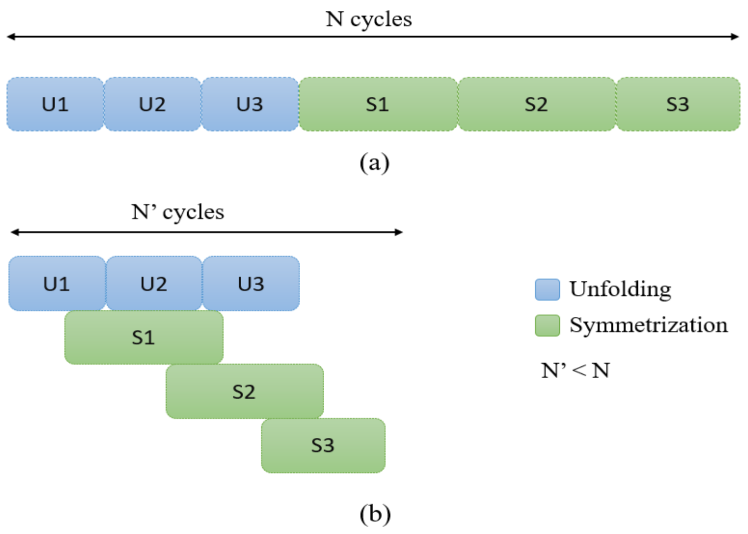
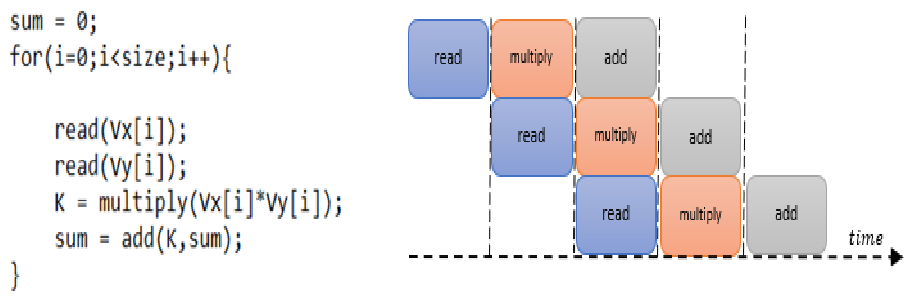


Figure 6. Dataflow pipelining: (**a**) without dataflow; and (**b**) with dataflow.



```
sum = 0;
for(i=0;i<size;i++){

    read(Vx[i]);
    read(Vy[i]);
    K = multiply(Vx[i]*Vy[i]);
    sum = add(K,sum);
}
```

Figure 7. Pipeline directive applied on vector multiplication.

5. FPGA Implementation Results and Assessment

5.1. Implementation Results

Figure 8 shows the normalized speedup and reduction in power consumption while assessing different approximate computing techniques. The latter are applied one-by-one resulting in eight different FPGA implementations. Each implementation is compared to the exact implementation where the time latency (L) is recorded as:

$$L = N \times \frac{1}{f_{max}} \quad (5)$$

where N is the number of clock cycles and f_{max} is the maximum operating frequency. As for the power consumption, a vector-based method was adopted as it provides the power consumption related to the processing under a defined testbench. The method involves generating a "saif" file via post-implementation functional and timing simulations.

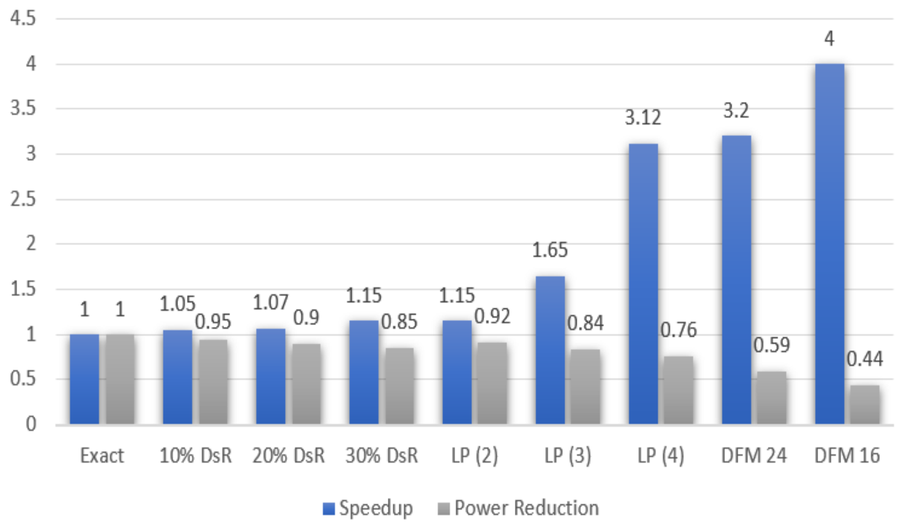


Figure 8. Speedup and power consumption reduction under different ACTs.

Using the obtained results in Figure 8, a cross-layer Approximate SVM implementation was performed where the adopted techniques are: 10% dataset reduction, loop perforation with $sf = 2$ and 24-bit DFM. Moreover, the implementation details was recorded with the performance booster ON and OFF to differentiate between the gain due to approximate computing techniques with and without HLS optimization directives. Table 2 summarizes the performance profile for the FPGA implementations based on the architecture in Figure 2. The Exact SVM is based on the architecture presented in [21]. The boosted Approximate

SVM corresponds to the Approximate SVM where the "Performance Booster" block is activated (See Figure 2), i.e., with HLS optimization directives. The reduction is calculated as:

$$Reduction(\%) = 100 - (\frac{I_{approx}}{I_{exact}} \times 100) \tag{6}$$

where I_{approx} and I_{exact} are the implementation element (FF, DSP, LUT, etc.) of the Approximate (or boosted approximate) and Exact SVM, respectively. As for the energy per classification, it is calculated using the equation:

$$E = P \times T \tag{7}$$

where T is the time latency and P is the dynamic power consumption reported in Vivado.

Table 2. FPGA performance profile of Exact and Approximate SVM.

	FF	LUT	DSP	BRAM	SRL	Time Latency (s)	Power Consumption (W)	Energy per Classification (J)	Classification Accuracy (%)
Exact SVM	37057	42261	475	297	1060	2.4	6.3	15.12	90
Approximate SVM	17,187	25,558	283	291	202	0.91	3.12	2.83	86
Boosted Approximate SVM	17,197	25,588	284	292	203	0.75	3.2	2.4	86
Approximate to Exact Reduction	53.62%	39.5%	40.4%	2.02%	80.94%	2.64×	50.4%	81.28%	−4%
Boosted Approximate to Exact Reduction	53.59%	39.45%	40.21%	1.68%	80.84%	3.2×	49.2%	84.12%	−4%

5.2. Implementation Assessment

The obtained results presented in Tables 1 and 2 and Figure 8 demonstrate the effectiveness of using approximate computing techniques to reduce the hardware resources utilization, time latency and power consumption of the FPGA implementation of the tensorial SVM. Such reductions are accompanied by an accuracy loss that varies between 0% and 10%. Another set of remarks can be noticed:

- In general, loop perforation achieves lower latency and power consumption compared to dataset reduction with a comparable accuracy loss. This can be justified since the SVD computation block is among the most complex blocks of the tensorial SVM, as reported in [4].
- The transition to fixed-point representation results in the lowest latency and power consumption compared to other methods. This is expected due to the reduced complexity of the arithmetic operations based on fixed-point representation. This can be seen in the reduced number of required DSPs between the exact and approximate implementations. However, this comes at the expense of high accuracy loss; for example, the use of a 16-bit fixed-point led to a 15% accuracy loss for the target application.
- The number of used BRAMs is high since we are not using any external DRAM for memory read/write operations. The range of the number of LUT and DSPs is expected due to the level of parallelism introduced using HLS directives. For the target FPGA, this is not a problem as long as we obtained a relatively reduced time latency and power consumption in the case of Approximate SVM.
- Using "cross-layer" approximate computing: with an accuracy degradation of 4%, the Approximate SVM requires about 43% less hardware resources and classifies an unseen sample 2.64× faster while consuming 50% less power compared to its exact counterpart.
- The accuracy loss due to the use of "cross-layer" approximate computing is not the sum of the losses obtained for each single approximate technique. This is evident in the final results presented in Table 2.

- The use of ACTs shows a remarkable reduction in the energy per classification up to 82%, since such techniques affect both the time latency and power consumption of the TSVM, as shown in Table 2.
- Applying the adopted HLS optimization directives offered an additional speedup gain to the Approximate SVM in terms of speedup up to 3.2× accompanied with 84% less energy per classification. This added a negligible overhead less than 1% increase in the hardware resources and power consumption. This is expected due to the fact that pipelining offers a reduction in the number of clock cycles while increasing the resources/power consumption. However, such increase is compensated by the dataflow directive that allows resource sharing, providing an enhanced overall implementation.

6. Conclusions

This paper presents the first FPGA implementation using High-Level Synthesis of an approximate tensorial SVM classifier. An accuracy of 86% was obtained with a speedup of 3.2× and 49% power consumption reduction resulting in up to 82% reduction in the energy per classification. Such results were achieved by utilizing the concept of cross-layer approximate computing. Combining several algorithmic level approximate computing techniques demonstrated their efficiency in optimizing the proposed embedded machine learning implementation. Moreover, specific design choices (e.g., using pipe-lined architecture) could boost the implementation performance with the help of Vivado HLS directives. The obtained reduction factor in hardware resources, time latency, and power consumption through applying ACTs paves the way towards applying such techniques on the RTL HDL design of the TSVM presented in [4] where similar reductions are expected. Such promising results motivate the exploration of different types of ACTs such as circuit-level techniques or the use of neural networks to further reduce the impact of computationally expensive singular value decomposition, as it represents about 70% of the whole implementation.

Author Contributions: Conceptualization, A.I.; methodology, H.Y. and A.I.; software, H.Y.; validation, M.V., A.I. and M.R.; investigation, H.Y. and A.I.; data curation, H.Y.; writing—original draft preparation, H.Y.; writing—review and editing, A.I., M.V. and M.R.; visualization, H.Y.; supervision, A.I., M.V. and M.R.; and funding acquisition, M.V. All authors read and agreed to the published version of the manuscript.

Funding: This research was funded by "TACTIle feedback enriched virtual interaction through virtual realITY and beyond (Tactility)": EU H2020, Topic ICT-25-2018-2020, RIA, Proposal ID 856718.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACT	Approximate Computing Technique
ARM	Advanced RISC Machines
ASIC	Application-Specific Integrated Circuits
FPGA	Field Programmable Gate Array
GPP	General Purpose Processor
HLS	High Level Synthesis
PDP	Power Delay Product
RCA	Ripple Carry Adder
SVD	Singular Value Decomposition
SVM	Support Vector Machine

References

1. Nayak, J.; Naik, B.; Behera, H.S. A Comprehensive Survey on Support Vector Machine in Data Mining Tasks: Applications & Challenges. *IJDTA* **2015**, *8*, 169–186. [CrossRef]
2. Afifi, S.M.; GholamHosseini, H.; Sinha, R. Hardware Implementations of SVM on FPGA: A State-of-the-Art Review of Current Practice. *Int. J. Innov.* **2015**, *2*, 2348–7968.
3. Magno, M.; Ibrahim, A.; Pullini, A.; Valle, M.; Benini, L. Energy Efficient System for Tactile Data Decoding Using an Ultra-Low Power Parallel Platform. In *2017 New Generation of CAS (NGCAS)*; IEEE: Genova, Italy, 2017; pp. 17–20. [CrossRef]
4. Ibrahim, A.; Valle, M. Real-Time Embedded Machine Learning for Tensorial Tactile Data Processing. *IEEE Trans. Circuits Syst. Regul. Pap.* **2018**, *65*, 3897–3906. [CrossRef]
5. Hussain, H.M.; Benkrid, K.; Seker, H. Reconfiguration-based implementation of SVM classifier on FPGA for Classifying Microarray data. In Proceedings of the 2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Osaka, Japan, 3–7 July 2013; pp. 3058–3061. [CrossRef]
6. Hussain, H.; Benkrid, K.; Şeker, H. Novel dynamic partial reconfiguration implementations of the support vector machine classifier on FPGA. *Turk. J. Electr. Eng. Comput. Sci.* **2016**, *24*, 3371–3387. [CrossRef]
7. Mandal, B.; Sarma, M.P.; Sarma, K.K. *Implementation of Systolic Array Based SVM Classifier Using Multiplierless Kernel*; 2014; pp. 35–39. Available online: http://www.wseas.us/e-library/conferences/2014/Brasov/ACMOS/ACMOS-46.pdf (accessed on 5 November 2020).
8. Vranjković, V.S.; Struharik, R.J.R.; Novak, L.A. Reconfigurable Hardware for Machine Learning Applications. *J. Circuits Syst. Comput.* **2015**, *24*, 1550064. [CrossRef]
9. Gastaldo, P.; Pinna, L.; Seminara, L.; Valle, M.; Zunino, R. Computational Intelligence Techniques for Tactile Sensing Systems. *Sensors* **2014**, *14*, 10952–10976. [CrossRef] [PubMed]
10. Sidiropoulos, N.D.; De Lathauwer, L.; Fu, X.; Huang, K.; Papalexakis, E.E.; Faloutsos, C. Tensor Decomposition for Signal Processing and Machine Learning. *IEEE Trans. Signal Process.* **2017**, *65*, 3551–3582. [CrossRef]
11. Signoretto, M.; De Lathauwer, L.; Suykens, J.A. A kernel-based framework to tensorial data analysis. *Neural Netw.* **2011**, *24*, 861–874. [CrossRef] [PubMed]
12. Mittal, S. A Survey of Techniques for Approximate Computing. *ACM Comput. Surv.* **2016**, *48*, 1–33. [CrossRef]
13. Van Leussen, M.; Huisken, J.; Wang, L.; Jiao, H.; Gyvez, J.P.D. Reconfigurable Support Vector Machine Classifier with Approximate Computing. In Proceedings of the 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Bochum, Germany, 3–5 July 2017; pp. 13–18. [CrossRef]
14. Wu, Y.; Yang, X.; Plaza, A.; Qiao, F.; Gao, L.; Zhang, B.; Cui, Y. Approximate Computing of Remotely Sensed Data: SVM Hyperspectral Image Classification as a Case Study. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2016**, *9*, 5806–5818. [CrossRef]
15. Zhou, Y.; Lin, J.; Wang, Z. Energy efficient SVM classifier using approximate computing. In Proceedings of the 2017 IEEE 12th International Conference on ASIC (ASICON), Guiyang, China, 25–28 October 2017; pp. 1045–1048. [CrossRef]
16. Ibrahim, A.; Osta, M.; Alameh, M.; Saleh, M.; Chible, H.; Valle, M. Approximate Computing Methods for Embedded Machine Learning. In Proceedings of the 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Bordeaux, France, 9–12 December 2018; pp. 845–848. [CrossRef]
17. Younes, H.; Ibrahim, A.; Rizk, M.; Valle, M. Algorithmic Level Approximate Computing for Machine Learning Classifiers. In Proceedings of the 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Genoa, Italy, 27–29 November 2019; pp. 113–114. [CrossRef]
18. Younes, H.; Ibrahim, A.; Rizk, M.; Valle, M. Data Oriented Approximate K-Nearest Neighbor Classifier for Touch Modality Recognition. In Proceedings of the 2019 15th Conference on Ph.D Research in Microelectronics and Electronics (PRIME), Lausanne, Switzerland, 15–18 July 2019; pp. 241–244. [CrossRef]
19. Nogues, E.; Menard, D.; Pelcat, M. Algorithmic-Level Approximate Computing Applied to Energy Efficient Hevc Decoding. *IEEE Trans. Emerg. Top. Comput.* **2016**, *7*, 5–17. [CrossRef]
20. Zhou, B.; Brent, R.; Kahn, M. Efficient one-sided Jacobi algorithms for singular value decomposition and the symmetric eigenproblem. In Proceedings of the 1st International Conference on Algorithms and Architectures for Parallel Processing, Brisbane, Australia, 19–21 April 1995; Volume 1, pp. 256–262. [CrossRef]
21. Osta, M.; Ibrahim, A.; Magno, M.; Eggimann, M.; Pullini, A.; Gastaldo, P.; Valle, M. An Energy Efficient System for Touch Modality Classification in Electronic Skin Applications. In Proceedings of the 2019 IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 26–29 May 2019; pp. 1–4. [CrossRef]
22. Chang, C.C.; Lin, C.J. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.* **2011**, *2*, 1–27. [CrossRef]
23. Xilinx, X. *Vivado Design Suite User Guide, High-Level Synthesis*; 2014. Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug902-vivado-high-level-synthesis.pdf (accessed on 5 November 2020).
24. Fares, H.; Seminara, L.; Ibrahim, A.; Franceschi, M.; Pinna, L.; Valle, M.; Dosen, S.; Farina, D. Distributed Sensing and Stimulation Systems for Sense of Touch Restoration in Prosthetics. In Proceedings of the 2017 New Generation of CAS (NGCAS), Genova, Italy, 6–9 September 2017; pp. 177–180. [CrossRef]