A Shallow Neural Network for Real-Time Embedded Machine Learning for Tensorial Tactile Data Processing

Hamoud Younes, *Student Member, IEEE,* Ali Ibrahim, *Member, IEEE,* Mostafa Rizk, *Member, IEEE,* and Maurizio Valle, *Senior Member, IEEE*

***Abstract*—This paper presents a novel hardware architecture and implementation of a Machine Learning (ML) method based on tensorial kernel approach dealing with multidimensional tensors. The architecture adopts shallow Neural Networks (NN) to compute the Singular Value Decomposition (SVD) of tensorial inputs. When implemented on an FPGA, the NN offers $324\times$ faster computations with reductions up to 58% and 67% in terms of hardware resources and power consumption respectively. When validated on a touch modality classification problem, the NN-based ML implementation has achieved a real-time operation while consuming about 88% less energy per classification than existing similar solutions. Such results offer the ability to deploy intelligence on resource-limited platform for energy-constrained applications.**

***Index Terms*—Embedded machine learning, real-time, tensorial kernel, tactile sensors, neural networks, singular value decomposition, FPGA.**

I. Introduction

TENSOR based learning techniques permit the effective exploitation of the structure of data used in various fields such vision (e.g. image recognition), neuroscience (e.g. MRI data), etc. Authors in [1] proposed a tensorial kernel that could be used for supervised tensor-based learning models while utilizing the structural information embodied in the data. When used with Support Vector Machine (SVM) algorithm, the tensorial kernel leads to better classification accuracy than the Gaussian-Radial Basis Function (RBF) and linear kernels in an image recognition task.

Gastaldo *et. al* have extended the tensorial kernel approach for tactile data processing in [2]. This approach has been adopted since it preserves the inherent tensorial structure of the data collected by tactile sensors. As an end result, the tensorial-based SVM achieves higher accuracy in classifying touch modalities compared to the Regularized Least Square (RLS) algorithm. In [3], the first FPGA implementation of the Support Vector Machine (SVM) algorithm based on tensorial kernel has been presented. Specifically, two implementations were provided: Cascaded and Parallel. The former failed to ensure real-time classification of touch (i.e in less than $400ms$ [4]), and the latter reported a relatively large hardware area and high power consumption of 1.14W. Such results were not acceptable for the application with limited power budget and area constraints [5].

In this paper, our main goal is to provide a new architecture and hardware implementation of the tensorial SVM (TSVM) aiming at reducing the hardware complexity and power consumption while keeping real-time operation. For this purpose, we analyzed the complexity of the tensorial SVM architecture to pin-out most computationally complex and demanding blocks. Fig. 1 illustrates the estimated number of operations required in each step of the tensorial SVM algorithm, where m and n are the dimensions of the unfolded matrix, N_c, N_t, and N_{sv} are the number of classes to be discriminated, the number of training tensors, and the number of support vectors respectively. It is evident that the Singular Value Decomposition (SVD) computation corresponds to about 96% of the overall algorithm. In [3], the one-sided Jacobi algorithm has been adopted for finding the singular vectors. Such algorithm involves a high number of arithmetic operations and requires several iterations to converge [6]. Thus, the main focus of the proposed new architecture is to find an alternative algorithm for SVD computation. This alternative should impose complexity reductions without affecting the classification accuracy of the tensorial SVM.

A Neural Network (NN) is one of the candidates for the SVD computation. The idea first surfaced in 1991 when Samardzija *et. al* proposed an artificial continuous time neural network to estimate the eignevectors and eigenvalues [7]. In [8], the convergence and computational complexity through computer simulations of such network are assessed. Another neural network has been presented in [9]. The network is

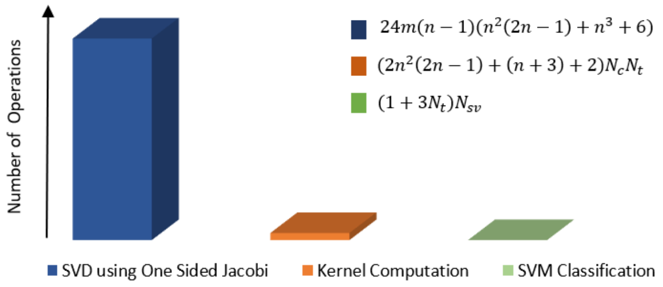

Fig. 1. Computational Complexity of the Tensorial SVM algorithm

Hamoud Younes, Ali Ibrahim, and Maurizio Valle are with the Department of Electrical, Electronic and Telecommunications Engineering and Naval Architecture, University of Genova, 16145 Genova, Italy (e-mail: hamoud.younes, ali.ibrahim@edu.unige.it; maurizio.valle@unige.it).

Hamoud Younes, Ali Ibrahim, and Mostafa Rizk are with the Department of Computer and Communication Engineering, Lebanese International University, Lebanon (e-mail: hamoud.younes, ali.ibrahim, mostafa.rizk@liu.edu.lb)

characterized by an order n-Ordinary Differential Equations (ODEs) leading to reduced dimensionality. Such neural network has evolved to further applications such as Principle Component Analysis (PCA) [10].

Triggered by the performance of neural networks in many domains [11] and the continuous quest for efficient designs specifically for resource-limited applications [12], a neural network based tensorial SVM architecture is proposed. The main contributions of this paper are summarized as follows:

- It presents a novel architecture for SVD computation using shallow neural networks. The architecture achieves 324× speedup with 58% and 67% reductions in the required hardware resources and power consumption respectively compared to the traditional one-sided Jacobi algorithm. Such reductions are obtained while providing a comparable performance in terms of Mean Squared Error (MSE) and Cosine Similarity (CS) metrics. Moreover, the proposed architecture is adequate for implementations on resource-limited platforms (e.g. Zynqberry [13]).
- It presents the first hardware implementation of a neural network based SVM featuring multidimensional tensorial inputs.
- It demonstrates the feasibility of the implemented system for real-time touch modality classification while consuming 6.28 mJ. The proposed cascade architecture achieves 131× classification speedup with a 39% and 50% resources and power reductions respectively compared to similar stat-of-the-art solution [3].
- It provides scalability assessment of the proposed SVD architecture. When used instead of the one-sided Jacobi computations in the tensorial SVM architecture, the neural network based SVM reports only 3% increase in the required FFs compared to 29% when the number of training tensors is doubled.

The rest of the paper is organized as follows: Section II presents an overview of the tensorial SVM for touch modality classification. Section III provides a review on the efficient existing SVD algorithms and their hardware implementations. It also reports the complexity of the proposed architecture compared to existing solutions. Section IV details the process of designing a neural network for SVD and its performance when tested on a tactile dataset. Section V provides the FPGA implementation and verification of the tensorial SVM based on SVD computation via shallow neural networks. Section VI presents a scalability study of the proposed architecture in terms of hardware resources and time latency. Section VII concludes the paper and illustrates on some observations.

II. SVM Classification based on Tensorial Kernel

A. Overview

A theoretical approach that extends kernel methods to tensor data has been presented in [14]. The framework allows the classification of an input tensor using SVM in 4 main steps:

- Tensor Unfolding: A tensor $\phi(I_1 \times I_2 \times I_3)$ is transformed into three matrices $X_1(I_1 \times I_2 I_3)$, $X_2(I_2 \times I_1 I_3)$ and $X_3(I_3 \times I_1 I_2)$.
- SVD Computation: The unfolded matrices are symmetrized into square matrices that can be written in the form:
$$X_1 = USV^T \qquad (1)$$
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 X_1.
- Kernel Computation: The tensorial kernel extended from the Gaussian kernel is computed using the function:
$$K(x,y) = \prod_{1}^{z} k^z(x,y) \qquad (2)$$
where k^z is the kernel factor defined as:
$$k(x,y) = \exp(\frac{-1}{2\sigma^2}(I_n - trace(Z^T Z))) \qquad (3)$$
where $Z = V_x^T V_y$, V_x and V_y represent the singular vectors of the unfolded matrix obtained during the inference and training phase respectively, and trace represents the sum of diagonal elements.
- Classification: Applying the SVM classification function expressed as:
$$\hat{y} = f_{SVM}(x) = \sum_{i}^{n} \beta_i K(x_i, x) + b \qquad (4)$$
where $\hat{y}$ is the predicted label of input tensor x, n is the number of training tensors, β_i are the coefficients obtained during training, and b is the bias.

B. Touch Modalities Classification

The tensorial SVM has been initially presented as an effective algorithm for touch modality classification in [14]. In this paper, three binary classification problems are used to test the accuracy of the proposed neural network based tensorial SVM. Specifically, the problems are:

- Problem A: ”brushing a paintbrush” versus ”rolling a washer”
- Problem B: ”brushing a paintbrush” versus ”sliding the finger”
- Problem C: ”sliding the finger” versus ”rolling a washer”

These modalities are derived from a tactile dataset that has been collected by 70 participants. Each participant performed the modality on both the horizontal and vertical axes of a 4×4 tactile sensor for a duration of 10 seconds. Thus each touch modality is represented by a tensor $\phi(4 \times 4 \times 30,000)$. However, such tensor size is reduced into $\phi(4 \times 4 \times 20)$ where 20 is the obtained number of samples using the data preprocessing algorithm (Algorithm 1) reported in section IV.

III. SVD Algorithms and Implementations

A. Literature Review

Singular value decomposition can be computed numerically through several methods such as: the Jacobi method, the QR method, and the one-sided Hestenes method [15]. For parallel implementations, computing the SVD using the Jacobi method is superior to other methods in terms of complexity and execution time [15]. Brent *et. al* have shown that two-dimensional systolic array could be used for implementing the Jacobi method [16]. In [17], the authors have presented various realization for the Jacobi SVD computation using Coordinate Rotation Digital Computer (CORDIC) [18]. The latter is adopted in majority of the existing hardware implementations of the Jacobi SVD method. For small matrix dimensions, an efficient implementation of SVD for the use in Multiple Input Multiple Output (MIMO) precoding and real-time signal processing has been presented in [19]. The implementation is based on CORDIC processors. For an arbitrary $m \times n$ matrix, Ibrahim *et. al* have presented an FPGA implementation with fixed-point arithmetic [20]. The implementation managed to compute the SVD of an 32×127 matrix in 13 ms while occupying 20% and 67% slice registers and LUTs respectively on a Virtex-6 FPGA. A fast and efficient FPGA implementation for computing the singular and eigen value decomposition based on a simplified CORDIC-like algorithm is presented in [21]. The implementation uses fixed-point arithmetic for sequential and parallel operations leading about 3× faster computation in an image denoising application compared to computations via an Intel CPU based PC. The authors in [22] used High-Level Synthesis (HLS) to model the one-sided Jacobi SVD computation on a Zedboard development board. For a 16×16 matrix, SVD computation takes around 1.1 seconds with a power consumption of 1.38W. Using CMOS 28-nm technology, Deng et.al proposed a hardware architecture for tensor SVD [23]. Compared with real-world CPU-based implementations, the architecture provides an average of 14× speed on various workloads.

Targeting the TSVM architecture in [3] where the one-sided Jacobi is identified as a performance bottleneck, the existing alternative implementations for SVD computation share several common challenges: (1) they operate only on square matrices. Thus, if the input matrix is rectangular, an additional complexity is added due to matrix symmetrization [23]. (2) if the implementation uses floating-point representation, the complexity is relatively high even for small matrix dimensions [24], and (3) depending on the required output precision, the algorithm might require additional iterations to converge [6]. Recently, a scalable SVD engine on FPGA has been introduced in [25] targeting these challenges. The proposed engine managed to compute the SVD of rectangular matrices using floating-point arithmetic. However, the implementation results show that a large number of DSPs is required for several matrix dimensions which has a direct impact on the power consumption of hardware implementations. Another noticeable observation is that the authors compared the SVD engine only to CPU-based SVD computations. In this paper, a new architecture for SVD computation based on shallow neural networks is proposed. The architecture offers the ability to operate on rectangular matrices (thus symmetrization is not needed, see Fig. 2) and utilizes floating-point arithmetic. As for convergence, the neural network training is usually performed offline on a high-end computing device. Thus, a network could be trained several times for any given amount of time to achieve top notch performance.

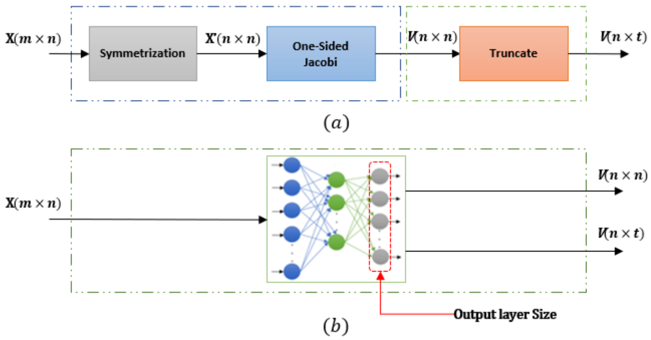

Fig. 2. SVD Computation using: (a) one-sided Jacobi, (b) Neural Network

B. Computational Complexity

In this section, we compare the complexity of the one-sided Jacobi algorithm with that of a shallow neural network in terms of the total number of operations. Consider a shallow neural network of one hidden layer of size H and an output layer of size O. For an input $A_{m\times n}$, the outputs of the hidden layer Y_h and the output layer Y_O are expressed respectively as:

$$Y_h = f_h(W_h.A + b_h) \tag{5}$$

$$Y_O = f_O(W_O.Y_h + b_O) \tag{6}$$

where W, b, and f represent the weight, bias, and activation function respectively. The output of each layer consists of matrix multiplication, addition, and activation operations. The number of operations for matrix multiplication and addition is expressed as:

$$N_h = H(2m \times n - 1) + H = 2H(m \times n) \tag{7}$$

Assuming that the activation function requires N_{Act} operations, the total number of operations in the hidden layers is expressed as:

$$N_h = 2H(m \times n) + N_{Acth} \tag{8}$$

The same can be applied to the output layer, thus the number of required operations is:

$$N_O = 2H \times O + N_{ActO} \tag{9}$$

Finally, the number of operations for the whole network could be expressed as:

$$N = N_h + N_O = 2H(m \times n + O) + N_{Acth} + N_{ActO} \tag{10}$$

To estimate N, suppose there exists an upper bound T such that $N \leq T$. T is an upper bound when both N_{Acth} and N_{ActO} correspond to the most complex activation function i.e. the tangent hyperbolic function ($tanh$). The latter is expressed as:

$$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{11}$$

To find the number of operations required for the term e^z, we referred to the function implementation in the IEEE-754 library in [26]. The implementation uses the Taylor expansion with an order 3 for floating-points, leading to a total of 16 operations. Thus the number of operations $N_{Acth} = 35H$. Similarly, $N_{ActO} = 35O$. For the network to output the right singular vectors V of an $m \times n$ matrix, the output layer size O is equal to n^2. This simplifies (10) to:

$$N <= (2H \times n)(m+n) + 35H + 35n^2 \tag{12}$$

Knowing that the number of operations for the one-sided Jacobi algorithm is (see Fig. 1):

$$Nj = 24m(n-1)[n^2(2n-1) + n^3 + 6] \tag{13}$$

through simulations, the values of m, n, and H are varied to compare (12) and (13). Fig. 3 plots the number of operations N_j and N required to compute the SVD of a matrix using one-sided Jacobi and a shallow neural network respectively. Generally, the comparison results are in favor of the neural network approach as shown in Fig. 3. The one-sided Jacobi is superior for very small dimensions such as 2×2 for $H > 21$. As the dimension starts to increase, the neural network requires significantly less number of operations for SVD computations. For instance, for $(m, n) = (20, 16)$ and $(m, n) = (4, 80)$ (these dimensions are often used for tensorial SVD implementations based on the one-sided Jacobi algorithm [3], [27]), computing the right singular vectors V using a shallow neural network requires less number of operations than using the one-sided Jacobi ($N < Nj$) for all values of $H \leq 70,000$ and $H \leq 800,000$ respectively. Such values of H are very large even for the largest existing neural networks.

IV. SVD using Neural Netwroks

A. Network Structure

A tactile tensor $\phi(4 \times 4 \times 20)$ is unfolded into three matrices $M(4 \times 80)$, $N(4 \times 80)$, and $P(20 \times 16)$. According to (1) each matrix could be decomposed into:

$$M_{4\times80} = U_{4\times80} \times \Sigma_{80\times80} \times V^T_{80\times80} \tag{14}$$

$$N_{4\times80} = U_{4\times80} \times \Sigma_{80\times80} \times V^T_{80\times80} \tag{15}$$

$$P_{20\times16} = U_{20\times16} \times \Sigma_{16\times16} \times V^T_{16\times16} \tag{16}$$

Authors in [14] and [28] reported that for tensor SVD, only a small number of the columns of V is required to obtain acceptable classification accuracy when embedded in SVM. Using the three touch modality problems reported in section II.B, the V matrices that resulted in the highest classification accuracy are: $V^T_{80\times4}$, $V^T_{80\times4}$, and $V^T_{16\times2}$.

Fig. 4(a) shows the proposed shallow neural network that is capable of computing the right singular vectors V. The network is composed of three fully connected layers: an input layer of size $m \times n$, a hidden layer of size H, and an output layer of size $O = n \times t$, where t is the selected number of columns from V. Thus, two neural networks are designed. one with an 80×4 output and the other with a 16×2 output.

The neural networks share two activation functions (f_h) and (f_O) defined as:

$$f_h(z) = max(\beta z, z) \tag{17}$$

$$f_O(z) = \begin{cases} -1 & z < -1 \\ 1 & z > 1 \\ z & otherwise \end{cases} \tag{18}$$

The function f_h shown in Fig. 4(c) is called leaky rectified linear unit (LeakyReLU) where β is a small constant used to keep negative values compared to the standard ReLU function. It is adopted for the hidden layer to preserve the sign of the neurons' output with low computational complexity compared to other activation functions (e.g. Sigmoid function). The

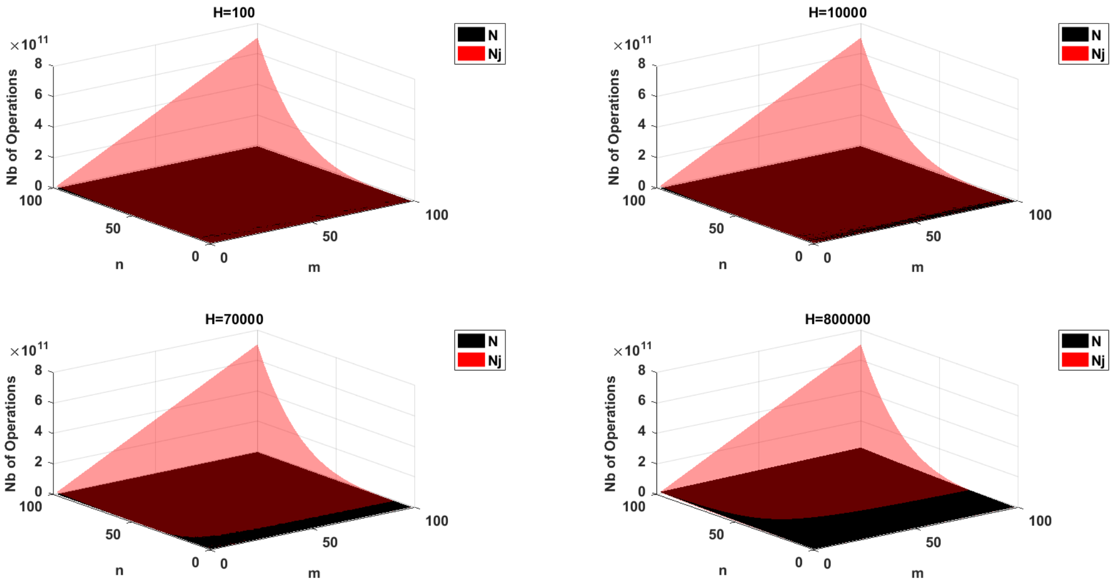

Fig. 3. Number of Operations required in one-sided Jacobi (Nj) and Shallow Neural Network (N), (m,n) are the matirx dimension and H is the hidden layer size.

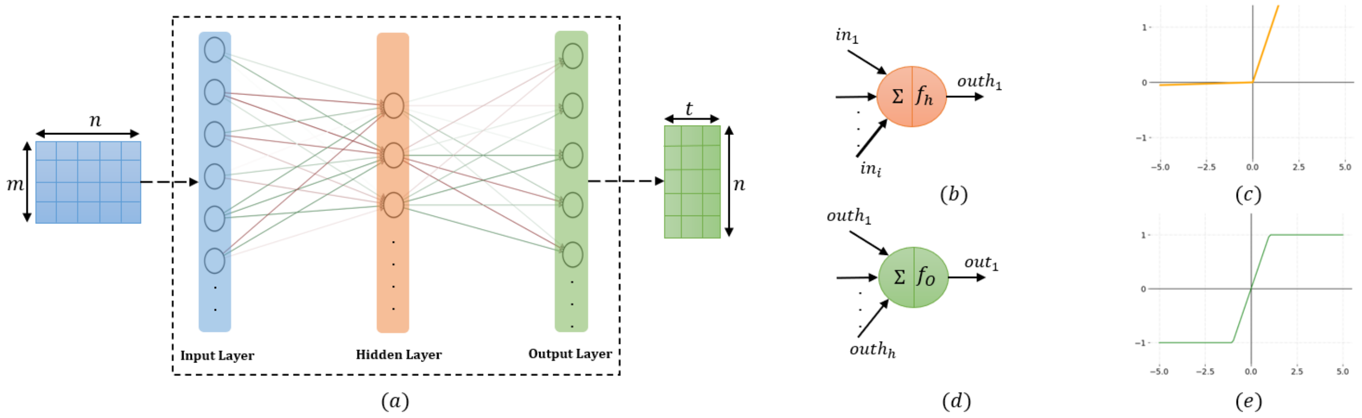

Fig. 4. Proposed Shallow Neural Network: (a) Overall Structure, (b) Hidden Layer Neuron, (c) LeakyReLU Activation Function, (d) Output Layer Neuron, (e) Approximate Hyperbolic Tangent Activation Function

function f_O shown in Fig. 4(e) is called hard hyperbolic tangent activation function [29]. It is used at the output layer to output the elements v_i of the V matrix in the range $[-1, 1]$ with a reduced computational complexity compared to the hyperbolic tangent function.

B. Network Training and Tuning

The chosen network model is trained using floating-point representation during both forward and backward propagation. The network is trained to export the right singular vectors V with the least possible error margin compared to exact computations obtained via MATLAB. The proposed network is a regression model that outputs singular vectors, for that the performance is determined based on two metrics: (1) Mean Squared Error (MSE) and (2) Cosine Similarity (CS). These metrics are defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(V_i - \hat{V}_i)^2 \tag{19}$$

$$CS = \frac{1}{n}\sum_{i=1}^{n}(\frac{V_i.\hat{V}_i}{||V_i|| \times ||\hat{V}_i||}) \tag{20}$$

where V is the matrix generated from the neural network and $\hat{V}$ is the one generated from applying the SVD using MATLAB software. Thus, the training aims at finding a network model that achieves the lowest MSE (i.e. the elements v_i of the V and $\hat{V}$ matrices have similar values) and highest CS (i.e. the vectors V_i of the V and $\hat{V}$ matrices have similar direction i.e CS tends to 1).

The proposed neural network is hand crafted and can be customized. The training process is used to tune the network structure (e.g. size of hidden layer H), parameters (e.g. weights), and hyperparameters (e.g. learning rate). During training, the weights and biases of the network are randomly initialized, then updated using one of the below optimizers. As for the hyperparameters, the following settings have been tested:

- H = [10, 20, ... 200]
- LeakyReLU β = [0.1, 0.01, 0.001]
- Learning rate = [0.1, 0.01, 0.001, ... 10^{-5}]
- Optimizer : [SGD, Adam, Adadelta, RMSprop]
- Batch size = [50, 100, 150]

The tactile dataset from [14] is used for training. However, some modifications have been applied based on the following:

- Some participants recordings are noisy (see Fig. 5(a)), thus their corresponding data has been removed from the training dataset.
- Since no particular indications were given to the participants in [14] about the pressure level, silent intervals (i.e. voltage readings from sensor taxels equals to zero, see Fig. 5(b)) are observed in the recordings. These silent intervals will not help the neural network to learn new patterns and thus are removed. Specifically, all reading outside the timing interval [3.5, 7] are omitted.

Algorithm 1 summarizes the pre-processing technique applied to the dataset. The algorithm truncates each modality from $10s$ to $3.5s$ resulting in a tensor $T'(4 \times 4 \times 10, 500)$. Afterwards, subsampling is applied to obtain 20 readings ($P = 20$) from

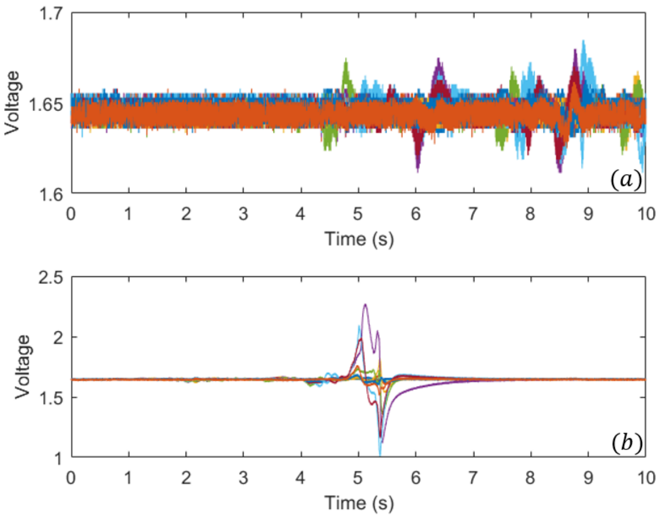

Fig. 5. Touch Modality with: (a) Noisy Readings, (b) Silent Intervals

the 10,500 resulting in a final tensor $\phi(4 \times 4 \times 20)$. After pre-processing, 4480 matrices of dimensions 4×80 and 20×16 have been derived. Then, their corresponding V matrices are generated using MATLAB. These matrices are divided into 80% for training, 10% for validation, and 10% for testing.

```
Algorithm 1: Pre-Processing Algorithm
  Input: Tensor T of size (:,:,S),
  Time Interval [a, b]
  Sampling parameter P
  Output: Sampled Tensor $\phi$ of size (:,:,P)
  Let $v1 \leftarrow a \times S/10$
  Let $v2 \leftarrow b \times S/10$
  Let $S' \leftarrow v2 - v1$
  Let $T'$ be a Tensor of size (:,:,S')
  Let $j = 0$
  for $i \leftarrow v1$ to $v2$ do
      $T'(:,:,j) \leftarrow T(:,:,i)$
      $j++$
  Let $k = 0$
  for $i \leftarrow 0$ to $P$ do
      $\phi(:,:,i) \leftarrow (P/S') * \sum_{i=k}^{S'/P+k} T'(:,:,i)$
      $k+= S'/P$
```

C. Network Performance

The neural network is coded in Python using Tensorflow and Keras libraries. Then, it is trained on an ASUS PC equipped with an NVIDIA GTX 1650 graphics card with 4GB VRAM. Fig. 6 shows the MSE and CS of the model with best achieved performance. The latter is obtained using the characteristics presented in Table I. one noticeable observation is that the size of the hidden layer differs for the two input dimensions. This is due to the fact that the network has to output 320 elements (80×4) for the input dimension (4×80) compared to 32 elements (16×2) for the input dimension (20×16), which justifies the longer training time required (higher number of epochs). However, the training can be shortened into 250 and 100 epochs for output dimensions (80×4) and (16×2) respectively.

TABLE I
BEST NEURAL NETWORK MODEL CHARACTERISTICS

Input Layer Size	20×16	4×80
Hidden Layer Size H	40	140
Output Layer Size	16×2	80×4
LeakyReLU β	0.01	
Learning Rate	0.001	
Batch Size	100	50
Epochs	400	1000

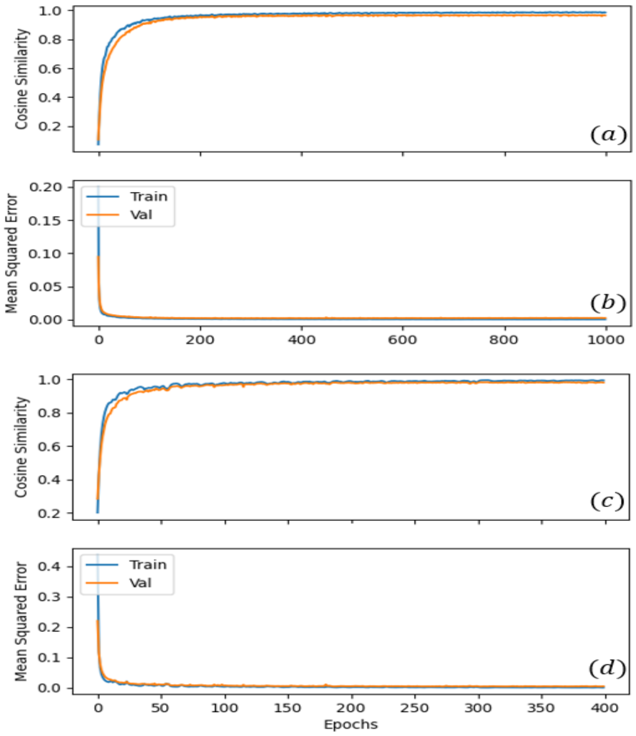

Fig. 6. Best Model Performance: (a) CS for $V(80 \times 4)$, (b) MSE for $V(80 \times 4)$, (a) CS for $V(16 \times 2)$, (b) MSE for $V(16 \times 2)$

The obtained performance is compared to that of computing the SVD using the one-sided Jacobi algorithm based on the architecture presented in [20]. According to the comparison shown in Table II, the proposed neural network is capable of computing the right singular vectors V while: (1) providing low MSE and high CS during training, validation, and testing, and (2) achieving comparable performance in terms of MSE and CS to the exact computation using the one-sided Jacobi. This is evident for both input dimensions 4×80 and 20×16.

V. HARDWARE IMPLEMENTATION AND VERIFICATION

This section presents the architecture and implementation details of the two shallow neural networks and the overall tensorial SVM. The latter is characterized by adopting these networks for SVD computation. Each architecture is modeled in C++, synthesized and implemented using Vivado HLS 2020.1 targeting Virtex-7 FPGA device operating at 100 MHz. For a credible power estimation, a post implementation functional and timing simulation using Vivado is performed to generate a Switching Activity Interchange File (SAIF). This file is used to obtain a vector-based power estimation post-routing.

For the rest of the paper, let NN1 and NN2 denote the neural networks with input dimensions 4×80 and 20×16 respectively.

TABLE II
BEST MODEL PERFORMANCE COMPARED TO ONE-SIDED JACOBI

Output Layer Size	16×2	80×4
Training MSE	9.6×10^{-4}	4×10^{-4}
Training CS	0.993	0.985
Validation MSE	9.71×10^{-4}	4.22×10^{-4}
Validation CS	0.979	0.964
Testing MSE	9.8×10^{-4}	4.3×10^{-4}
Testing MSE based on [20]	9.21×10^{-4}	3.88×10^{-4}
Testing CS	0.966	0.952
Testing CS based on [20]	≈ 1	≈ 1

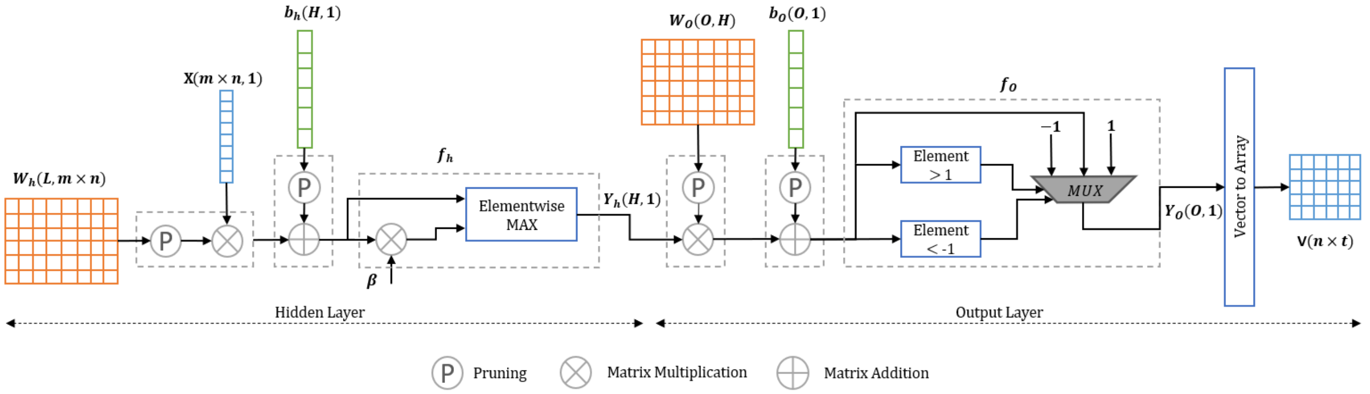

Fig. 7. Shallow Neural Network Architecture

A. FPGA Implementation of The Shallow Neural Network

Fig. 7 shows the architecture of the proposed shallow neural network. For an input X of size L (one of the unfolded matrices), it outputs the V matrix using sequential operations. The outputs Y_h and Y_O corresponds to the equations (5) and (6), where f_h and f_O are the LeakyReLU and the hard tangent hyperbolic activation functions respectively. The advantage of such architecture is that it allows the use of network pruning without any loss in performance (MSE/CS). Pruning is applied on matrix multiplication/addition by skipping operations where $W[i], b[i] \leq 10^{-4}$. Table III shows the implementation details for the SVD computation of a $4 \times 4 \times 20$ tensor (i.e. two NN1 to compute the SVD of the matrices M, N and one NN2 to compute the SVD of the matrix P) compared to the one-sided Jacobi based on the architecture presented in [3]. The obtained results show that using neural networks for SVD computations allows for a $324\times$ speedup with an average resources and power reductions of 58% and 67% respectively. Another observation is that the neural network architecture uses slightly more BRAMs. This is due to the fact that the weight and bias matrices obtained from network training are mapped into BRAMs and are not saved on an external memory. Knowing that the Virtex-7 FPGA is used for implementation to have a credible comparison with the state-of-the-art, the obtained results show that the proposed neural network for SVD computations is adequate to fit in a resource-limited platform such as the Zynqberry. This is not possible for the implementation of the one-sided Jacobi targeting large matrix dimensions.

TABLE III
IMPLEMENTATION RESULTS FOR TENSOR SVD COMPUTATIONS

Architecture	Neural Network	one-sided Jacobi
BRAM	102	88
DSP	32	105
FF	3714	29277
LUT	4905	43258
Time Latency	14.5 ms	4.7 s
Power Consumption	0.45W	1.35W

B. FPGA Implementation of The Neural Network based SVM

The neural networks NN1 and NN2 have been embedded into the cascade architecture of the tensorial SVM presented in [3]. The new NN-based TSVM architecture is presented in Fig. 8. The ”NN Memory” contains the weight and bias matrices of the designed neural networks. The ”SVM Memory” contains the singular vector training matrices. k_1, k_2, and k_3 are the three kernel factors obtained using (3).

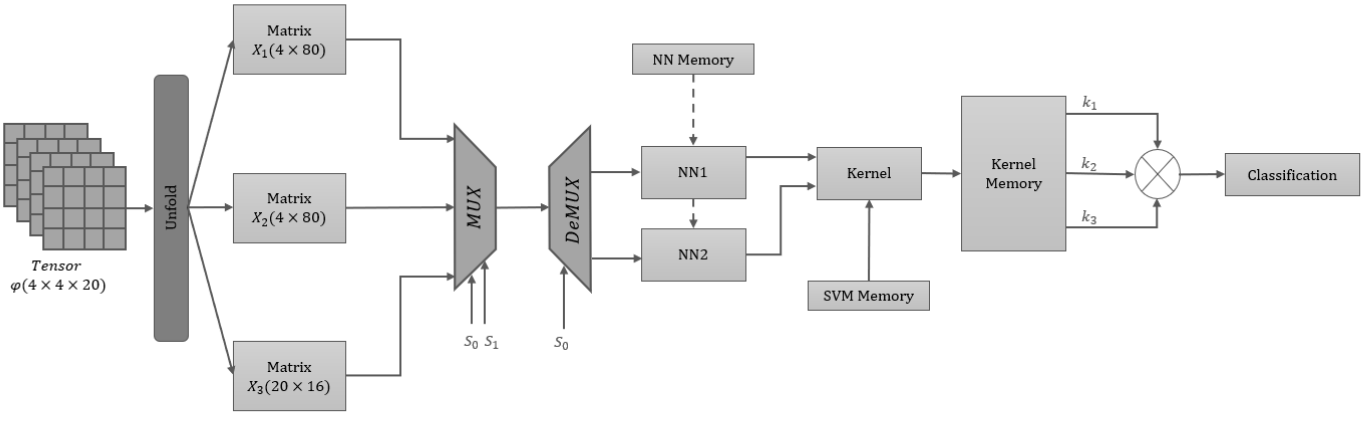

Fig. 8. Neural Network based SVM Cascade Architecture

TABLE IV
IMPLEMENTATION RESULTS FOR TENSORIAL SVM

Architecture	NN-TSVM	Jacobi-TSVM
BRAM	105	91
DSP	133	206
FF	11975	39047
LUT	20427	60100
Time Latency (ms)	36	4730
Energy per classification (mJ)	6.28	600

Table IV presents the implementation details of both the NN-based TSVM and Jacobi-based SVM for $N_t = 200$ and $N_c = 2$. The energy per classification is computed as $E = P \times T$ where P is the dynamic power consumption and T is the time latency. The NN-based TSVM and Jacobi-based TSVM recorded 0.9 W and 1.8 W respectively. Results show that replacing the one-sided Jacobi algorithm with a shallow neural network in the architecture of the TSVM leads to faster classification time up to 131×. The NN-based TSVM also requires 39% less average hardware resources with 50% reduced power consumption This leads to 88% reductions in the energy per classification factor. Two main observations could be noted: the proposed NN-based TSVM (1) is capable of real-time classification within 36 ms ($time \leq 400ms$ [4]), (2) achieves real-time classification using cascaded architecture, which was not possible using the Jacobi-based TSVM as reported in [3]. The latter has been the main reason for using the parallel architecture which has lead to high power consumption.

C. Performance Verification

The NN-based TSVM implementation is verfied using the three binary classification problems mentioned in Section II.B. Table V shows the classification accuracy of different TSVM architectures obtained by testing the implementation on a dataset with 30 testing samples. Using neural networks to compute the right singular vectors V provides approximate values compared to the exact one-sided Jacobi. However, this resulted in acceptable classification accuracy with only 3% loss at the worst. This is evident in the comparable MSE/CS of both architectures as presented in Table V.

VI. SCALABILITY OF NEURAL NETWORK BASED TSVM

In order to quantify the scalability of the NN-based TSVM hardware complexity (resources and time latency), two cases are assessed: (1) Scalability of the shallow neural network, and (2) Scalability of the NN-based TSVM.

TABLE V
TOUCH MODALITY CLASSIFICATION USING DIFFERENT TSVM ARCHITECTURES

Problem	Classification Accuracy (%)	
	NN-TSVM	Jacobi TSVM
A	90	90
B	83.3	86.6
C	80	83.3

A. Case 1:

The scalability of the neural network depends on the size of each layer and the activation function in use. Through Fig. 3, an insight about the number of operations with respect to the dimensions (i.e. m, n, and H) could be learned for a certain application. To assess the scalability of the proposed NN architecture, the output layer size is varied. Thus, we designed and synthesized another two neural networks that are capable of predicting the right singular vectors V without truncation i.e output layer size $O = n \times n$ instead of $O = n \times t$. Fig. 9 presents the variation of the hardware resources and time latency with respect to different hidden and output layer sizes. The obtained results are recorded when the network achieved a comparable MSE/CS to those reported in Table II. Analyzing the graphs leads to several observations:

- The number of required FFs and LUTs is not uniform (see Fig. 9(a),(b)). For instance, a similar number of FFs/LUTs is required for networks with 140 and 400 neurons in the hidden layer with the same output layer size. This could be justified with the pruned cascaded architecture where resources are shared for blocks with similar functionality.
- Memory requirements in terms of BRAMs starts to increase once reached an output layer size of 80×80 with 400 neurons in the hidden layer (see Fig. 9(c)). This is justified since the sizes of the weight and the bias matrices increase in such cases, which requires more storage memory.
- As shown in Fig. 9(d), regardless of the input/hidden/output layer size of the network, the number of DSPs is constant for the proposed architecture.
- The SVD computation time is relatively short until reaching a high output layer size as shown in Fig. 9(e). This is due to the longer operations required to perform matrix multiplication/addition. However, according to the comparison in Section III.B, this is faster than using the one-sided Jacobi as long as $H \leq 70,000$ ($H \leq 800,000$) for 20×16 (4×80) matrices.

The presented scalability assessment supports the use of these networks for SVD computations as an efficient solution especially for large matrix dimensions. Hence, the proposed idea could be extended into other applications via a two-stage approach as shown in Fig. 10:

- *Stage 1:* Unfold all the tensors ϕ_i in a dataset into 3 matrices. Then, find the V matrix for each of the unfolded matrices using MATLAB or other software. For the majority of the applications, a tensor has the same first two dimensions (e.g. image, touch modality) hence, two of the generated matrices will have the same dimension hence can be grouped in a subset A. The remaining matrix and its corresponding V matrix will be added to a subset B.
- *Stage 2:* For each of the subsets, a shallow neural network is to be designed. Start with random hyperparameters for the initial model, then tune it using the generated subset to reach the required MSE and CS. Once, the best model is found, the weight and bias matrices could be exported and used by the architecture in Fig. 7. For complexity tuning,

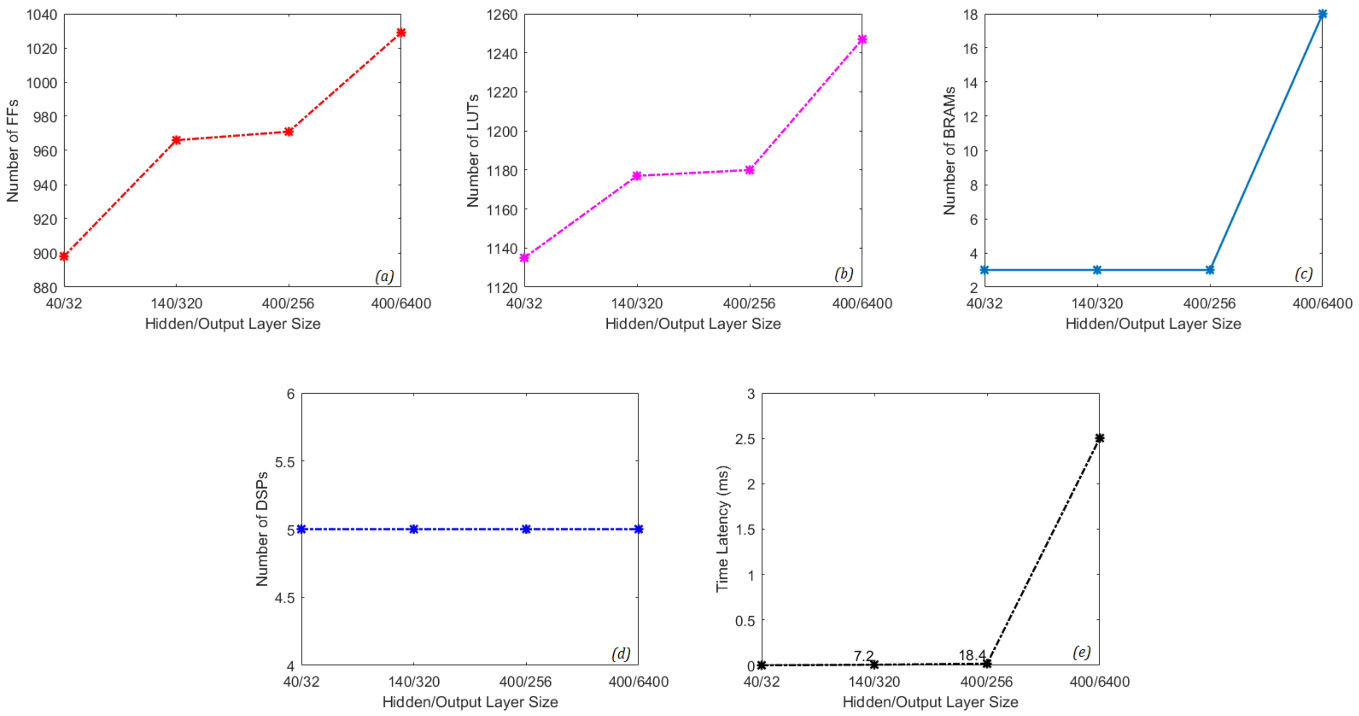

Fig. 9. Scalability of Shallow Neural Network for varying the hidden/output layers size

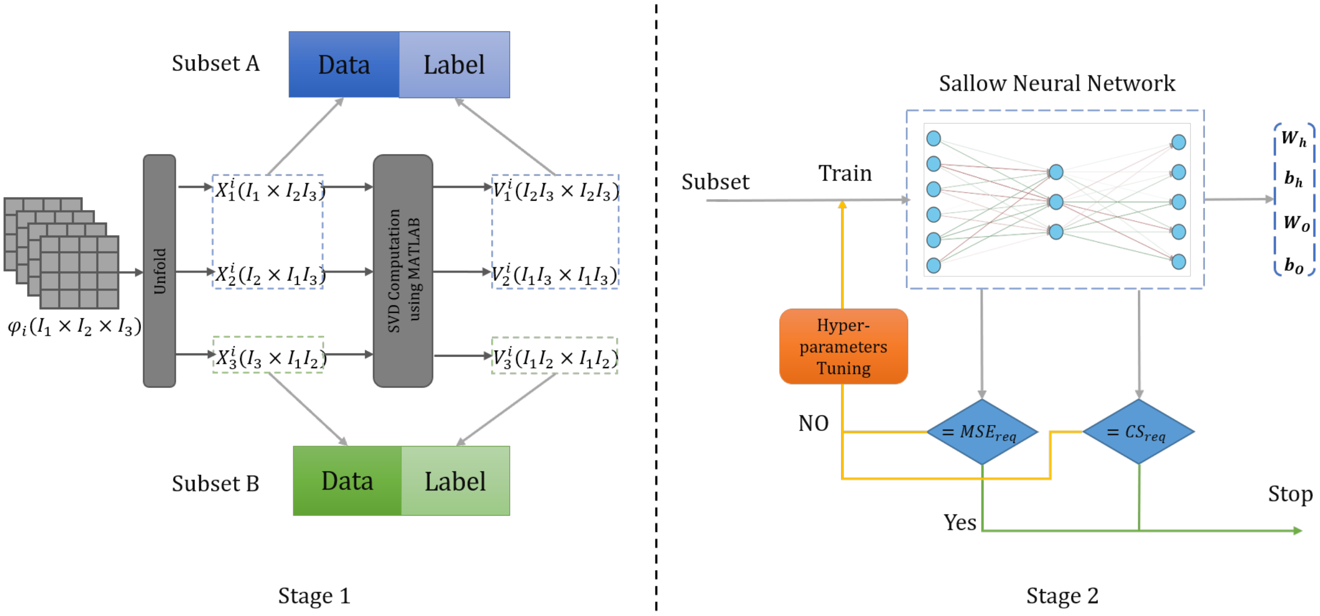
Subset A
Data
Label
$\varphi_i(I_1 \times I_2 \times I_3)$
Unfold
$X_1^i(I_1 \times I_2 I_3)$
$X_2^i(I_2 \times I_1 I_3)$
$X_3^i(I_3 \times I_1 I_2)$
SVD Computation using MATLAB
$V_1^i(I_2 I_3 \times I_2 I_3)$
$V_2^i(I_1 I_3 \times I_1 I_3)$
$V_3^i(I_1 I_2 \times I_1 I_2)$
Subset B
Data
Label
Stage 1
Sallow Neural Network
Subset
Train
W_h
b_h
W_o
b_o
Hyper-parameters Tuning
NO
$= MSE_{req}$
$= CS_{req}$
Yes
Stop
Stage 2

Fig. 10. Stage 1:SVD Computation Approach via Shallow Neural Network

one could modify the pruning rule while preserving the required performance metric imposed by the application.

B. Case 2:

To study the scalability of the proposed NN-based TSVM, the number of training tensors has been varied between 200 and 900 and the implementation requirements are recorded once the NN-based TSVM recorded a comparable accuracy to the one presented in Table V. According to the results obtained in Fig. 11:

- The required hardware resources (FFs, LUTs, BRAMs) are slightly increased with the increase of the number of training tensors. In case of BRAMs, a steeper slope is observed which is due to the adoption of NN that requires the storage of weight and bias matrices.
- The number of required DSPs is contsant for each size of training tensors.

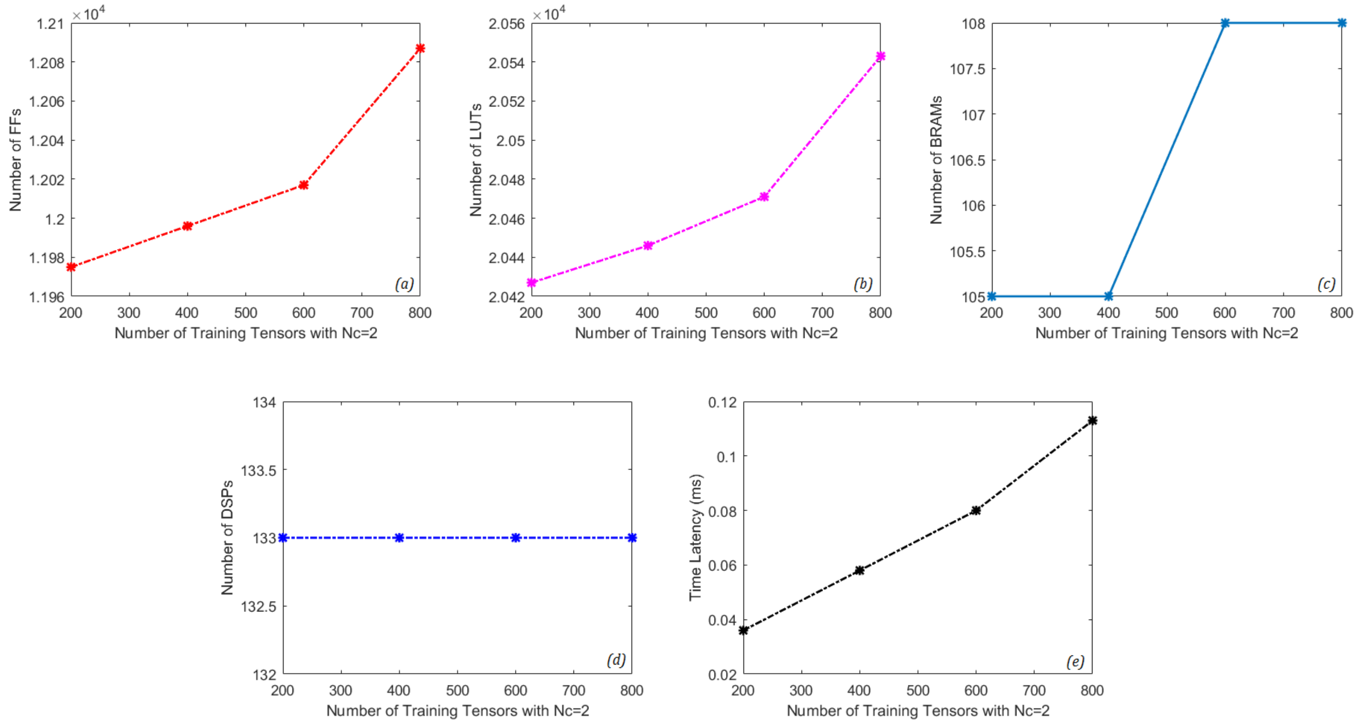

Fig. 11. Scalability of NN-based TSVM for binary classification (Nc=2) and variable number of training tensors

- The proposed implementation is capable of real-time classification even after $4.5\times$ increase in the number of training tensors.

Compared to the scalability study of the Jacobi-based TSVM presented in [30], the NN-based TSVM complexity shows a smaller slope for resource increase. For example, the Jacobi-based TSVM requires 29% increase in the number of FFs when the number of training tensors N_t is doubled. Using the NN-based TSVM, only 3% increase in FFs is noticed. This is mainly due to two reasons: (1) the neural network requires significantly less resources than that of the one-sided Jacobi. (2) the NN-based TSVM is a cascaded implementation i.e. blocks are being re-used for implementation while increasing the time latency. In [30], the architecture is based on parallel computation due to their time constraint of real-time classification. The latter is assured using the proposed cascaded architecture for all of training tensors sizes.

The importance of the presented work lies in the ability to scale such architecture for processing larger number of samples while respecting the constraints of the application. When scaled up, the designed NN-TSVM could enable intelligence on smaller platforms (e.g. Zynqberry) if two issues are tackled. The first issue is reducing the number of DSPs: this could be achieved by using some approximate computing techniques [31] or using LUTs-only custom core for matrix operations. The second issue is reducing the number of BRAMs: this could be achieved by further pruning of the weight/bias matrices as long as the application performance is not highly affected. Another method is to offload these matrices completely to external DRAM. This imposes additional timing overhead. However, authors in [32] have presented a strategy to overcome such design challenge.

VII. Conclusion

This paper introduced a shallow neural network architecture for the SVD computation of tensorial inputs. The architecture achieves comparable performance to the state-of-art solutions while imposing significant reductions in the implementation requirements. Once embedded in the SVM architecture, the NN-based TSVM is capable of delivering faster touch modality classification time up to $131\times$ using a cascade architecture. The latter is characterized by a 39% and 88% decrease in the resource and energy per classification respectively compared to the architecture presented in [3] targeting the same application. Moreover, the proposed NN-based SVM obeys the constraints imposed by the tactile data processing application e.g. small size, real-time response, and low power consumption. The encouraging scalability results present the first effective trial for designing an efficient embedded processing unit for an e-skin. A unit that is capable of delivering real-time performance with relatively acceptable power consumption without the need for high performance platform or multi-core devices.

Acknowledgement

The authors acknowledge partial financial support from TACTIle feedback enriched virtual interaction through virtual realITY and beyond (TACTILITY) project: EU H2020, Topic ICT-25-2018-2020, RIA, Proposal ID 856718.

References

[1] M. Signoretto, L. De Lathauwer, and J. A. Suykens, "A kernel-based framework to tensorial data analysis," *Neural Networks*, vol. 24, pp. 861–874, Oct. 2011.

[2] P. Gastaldo, L. Pinna, L. Seminara, M. Valle, and R. Zunino, "A Tensor-Based Pattern-Recognition Framework for the Interpretation of Touch Modality in Artificial Skin Systems," *IEEE Sensors Journal*, vol. 14, pp. 2216–2225, July 2014.

[3] A. Ibrahim and M. Valle, "Real-Time Embedded Machine Learning for Tensorial Tactile Data Processing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, pp. 3897–3906, Nov. 2018.

[4] P. P. Lele, D. C. Sinclair, and G. Weddell, "The reaction time to touch," *The Journal of Physiology*, vol. 123, pp. 187–203, Jan. 1954.

[5] H. Fares, L. Seminara, A. Ibrahim, M. Franceschi, L. Pinna, M. Valle, S. Dosen, and D. Farina, "Distributed Sensing and Stimulation Systems for Sense of Touch Restoration in Prosthetics," in *2017 New Generation of CAS (NGCAS)*, (Genova, Italy), pp. 177–180, IEEE, Sept. 2017.

[6] B. Zhou, R. Brent, and M. Kahn, "Efficient one-sided Jacobi algorithms for singular value decomposition and the symmetric eigenproblem," in *Proceedings 1st International Conference on Algorithms and Architectures for Parallel Processing*, vol. 1, (Brisbane, Qld., Australia), pp. 256–262, IEEE, 1995.

[7] N. Samardzija and R. L. Waterland, "A neural network for computing eigenvectors and eigenvalues," *Biol. Cybern.*, vol. 65, pp. 211–214, Aug. 1991.

[8] Z. Yi, Y. Fu, and H. J. Tang, "Neural networks based approach for computing eigenvectors and eigenvalues of symmetric matrix," *Computers & Mathematics with Applications*, vol. 47, pp. 1155–1164, Apr. 2004.

[9] Y. Tang and J. Li, "Another neural network based approach for computing eigenvalues and eigenvectors of real skew-symmetric matrices," *Computers & Mathematics with Applications*, vol. 60, pp. 1385–1392, Sept. 2010.

[10] J. Qiu, H. Wang, J. Lu, B. Zhang, and K.-L. Du, "Neural Network Implementations for PCA and ItsExtensions," *ISRN Artificial Intelligence*, vol. 2012, pp. 1–19, 2012.

[11] Z. Li, W. Yang, S. Peng, and F. Liu, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," *arXiv:2004.02806 [cs, eess]*, Apr. 2020. arXiv: 2004.02806.

[12] M. P. Véstias, "A Survey of Convolutional Neural Networks on Edge with Reconfigurable Computing," *Algorithms*, vol. 12, p. 154, July 2019.

[13] *TE0726 Resources - Public Docs - Trenz Electronic Wiki.*

[14] P. Gastaldo, L. Pinna, L. Seminara, M. Valle, and R. Zunino, "Computational Intelligence Techniques for Tactile Sensing Systems," *Sensors*, vol. 14, pp. 10952–10976, June 2014.

[15] B. Yang and J. F. Böhme, "Reducing the Computations of the Singular Value Decomposition Array Given by Brent and Luk," *SIAM J. Matrix Anal. & Appl.*, vol. 12, pp. 713–725, Oct. 1991.

[16] R. P. Brent and F. T. Luk, "The Solution of Singular-Value and Symmetric Eigenvalue Problems on Multiprocessor Arrays," *SIAM J. Sci. and Stat. Comput.*, vol. 6, pp. 69–84, Jan. 1985.

[17] J. R. Cavallaro and F. T. Luk, "Architectures For A Cordic SVD Processor," (San Diego), p. 45, Mar. 1986.

[18] J.-M. Delosme, "CORDIC Algorithms: Theory And Extensions," (San Diego), p. 131, Nov. 1989.

[19] D. Milford and M. Sandell, "Singular value decomposition using an array of CORDIC processors," *Signal Processing*, vol. 102, pp. 163–170, Sept. 2014.

[20] A. Ibrahim, M. Valle, L. Noli, and H. Chible, "Singular value decomposition FPGA implementation for tactile data processing," in *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, (Grenoble, France), pp. 1–4, IEEE, June 2015.

[21] S. Zhang, X. Tian, C. Xiong, J. Tian, and D. Ming, "Fast Implementation for the Singular Value and Eigenvalue Decomposition Based on FPGA," *Chinese Journal of Electronics*, vol. 26, pp. 132–136, Jan. 2017.

[22] T. Jiang, F. Xie, S. Yuan, S. Yao, K. Wu, and X. Wang, "Implementation of Matrix SVD Decomposition Module for Subspace Channel Estimation," in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, (Chengdu, China), pp. 1096–1102, IEEE, Mar. 2019.

[23] C. Deng, M. Yin, X.-Y. Liu, X. Wang, and B. Yuan, "High-performance Hardware Architecture for Tensor Singular Value Decomposition: Invited Paper," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, (Westminster, CO, USA), pp. 1–6, IEEE, Nov. 2019.

[24] A. Ibrahim, M. Valle, L. Noli, and H. Chible, "FPGA implementation of fixed point CORDIC-SVD for E-skin systems," in *2015 11th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, (Glasgow, United Kingdom), pp. 318–321, IEEE, June 2015.

[25] Y. Wang, J.-J. Lee, Y. Ding, and P. Li, "A Scalable FPGA Engine for Parallel Acceleration of Singular Value Decomposition," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*, (Santa Clara, CA, USA), pp. 370–376, IEEE, Mar. 2020.

[26] H. Bui and S. Tahar, "Design and synthesis of an IEEE-754 exponential function," in *Engineering Solutions for the Next Millennium. 1999 IEEE Canadian Conference on Electrical and Computer Engineering (Cat. No.99TH8411)*, vol. 1, (Edmonton, Alta., Canada), pp. 450–455, IEEE, 1999.

[27] M. Osta, M. Alameh, H. Younes, A. Ibrahim, and M. Valle, "Energy Efficient Implementation of Machine Learning Algorithms on Hardware Platforms," (Genova, Italy), Nov. 2019.

[28] M. Osta, A. Ibrahim, M. Magno, M. Eggimann, A. Pullini, P. Gastaldo, and M. Valle, "An Energy Efficient System for Touch Modality Classification in Electronic Skin Applications," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, (Sapporo, Japan), pp. 1–4, IEEE, May 2019.

[29] C. Gulcehre, M. Moczulski, M. Denil, and Y. Bengio, "Noisy Activation Functions," *arXiv:1603.00391 [cs, stat]*, Apr. 2016. arXiv: 1603.00391.

[30] A. Ibrahim, P. Gastaldo, H. Chible, and M. Valle, "Real-Time Digital Signal Processing Based on FPGAs for Electronic Skin Implementation †," *Sensors*, vol. 17, p. 558, Mar. 2017.

[31] S. Mittal, "A Survey of Techniques for Approximate Computing," *ACM Computing Surveys*, vol. 48, pp. 1–33, Mar. 2016.

[32] J. Vieira, R. P. Duarte, and H. C. Neto, "kNN-STUFF: kNN STreaming Unit for Fpgas," *IEEE Access*, vol. 7, pp. 170864–170877, 2019.

Hamoud Younes He received the B.S and M.S. degrees in Computer and Communication Engineering from the Lebanese International University, in 2012 and 2015 respectively. Currently, he is a Ph.D. student at the Department of Electric, Electronic, Telecommunication Engineering and Naval Architecture, University of Genoa. His research interests involve embedded electronic systems, FPGA implementation, embedded machine and deep learning, approximate computing, and energy efficient embedded computing.

Ali Ibrahim received the M.S. degree in industrial control from the Doctoral School of Sciences and Technologies, Lebanese University, in 2009, and the dual Ph.D. degrees in electronic and computer engineering and robotics and telecommunications from the University of Genova and the Lebanese University in 2016. He has been a Post-Doctoral Researcher with the Department of Electric, Electronic, Telecommunication Engineering and Naval Architecture, University of Genova from 2016 to 2018. Currently, he is an assistant professor in the department of Electrical and Electronics Engineering at the Lebanese international University in Lebanon and also an associate researcher Department of Electric, Electronic, Telecommunication Engineering and Naval Architecture, University of Genoa. His research interests involve Embedded machine learning, FPGA implementation, and interface electronics for electronic skin systems, approximate computing, and techniques and methods for energy efficient embedded computing.

Mostafa Rizk received the Maitrise degree in electronics, the M.Sc. degree in biomedical physics, and the M.Sc. degree in signal, telecom, image, and speech from Lebanese University, Beirut, Lebanon, in 2007, 2008, and 2010, respectively, the Ph.D. degree in sciences and technologies of information from Telecom Bretagne, Brest, France, in 2014, and the Ph.D. degree in electronics and communication from Lebanese University in 2015. He has been a Post-Doctoral Researcher with the University of Southern Brittany, Lorient, France, and with the Lab-STICC Laboratory CNRS, Lorient. He is currently an Assistant Professor with Lebanese International University, Beirut, and an Associate Researcher with IMT Atlantique, Brest, France. His current research interests include hardware/software implementations and digital circuit design, network-on-chip design, and new MPSoC architectures based on emerging nonvolatile memory technologies.

Maurizio Valle (SM'16) received the M.S. degree in electronic engineering and the Ph.D. degree in electronic and computer science engineering from the University of Genova, Italy, in 1985 and 1990, respectively. From 1992 to 2006, he was an Assistant Professor. Since 2007, he has been an Associate Professor in electronic engineering with the Department of Electrical, Electronic and Telecommunications Engineering and Naval Architecture, University of Genova. He has co-authored over 200 papers on international scientific journals and conference proceedings and of the book Robotic Tactile Sensing Technologies and System (Springer Science + Business Media, Dordrecht, pp. 1–248, 2013, ISBN: 978-94-007-0578-4). He has been/is in charge of many research contracts and projects funded at local, national, and European levels, and by Italian and foreign companies. His research interests include material integrated sensing systems, electronic tactile sensors and systems, microelectronic embedded systems, and wireless sensors networks. He was a Technical Program Committee Member of the IEEE 2011 and 2013–2016 PRIME Conferences. He has been a Guest Editor of the Special Issue on Robotic Sense of Touch in the IEEE TRANSACTIONS ON ROBOTICS (Volume 27, Issue 3, 2011) and a Guest Editor of the Special Issue on Tactile Sensors and Sensing Systems in Sensors (MDPI, Open Access Publishing, ISSN 1424-8220, December 15, 2013).