Tolerance of Siamese Networks (SNs) to Memory Errors: Analysis and Design

Tolerance of Siamese Networks (SNs) to Memory Errors: Analysis and Design

Ziheng Wang, *Student Member, IEEE*, Farzad Niknia, *Student Member, IEEE*, Shanshan Liu, *Member, IEEE*, Pedro Reviriego, *Senior Member, IEEE*, Paolo Montuschi, *Fellow, IEEE* and Fabrizio Lombardi, *Life Fellow, IEEE*

***!"#$%&'$*—**This paper considers memory errors in a Siamese Network (SN) through an extensive analysis and proposes two schemes (using a weight filter and a code) to provide efficient hardware solutions for error tolerance. Initially the impact of memory errors on the weights of the SN (stored as floating-point (FP) numbers) is analyzed; this shows that the degradation is mostly caused by outliers in weights. Two schemes are subsequently proposed. An analysis is pursued to establish the filter's bounds selection by the maximum/minimum values of the weight distributions, by which outliers can be removed from the operation of the SN. A code scheme for protecting the sign and exponent bits of each weight in an FP number, is also proposed; this code incurs in no memory overhead by utilizing the 4 least significant bits (LSB) to store parity bits. Simulation shows that the filter has a better performance for multi-bit errors correction (a reduction of 95.288% in changed predictions), while the code achieves superior results in single-bit errors correction (a reduction of 99.775% in changed predictions). The combined method that uses the two proposed schemes, retains their advantages, so adaptive to all scenarios; The ASIC-based FP designs of the SN using serial and hybrid implementations are also presented; these pipelined designs utilize a novel multi-layer perceptron (MLP) (as branch networks of the SN) that operates at a frequency of 681.2 MHz (at a 32nm technology node), so significantly higher than existing designs found in the technical literature. The proposed error-tolerant approaches also show advantages in overheads comparing with for example traditional error correction code (ECC). These error-tolerant MLP-based designs are well suited to hardware/power-constrained platforms.**

***()*+,- .+%/#—Siamese Network, Memory Error, Error Tolerance, VLSI Design, Floating point.**

I. Introduction

Siamese Networks (SNs) are a type of artificial neural networks (ANNs) consisting of two branches of weight-sharing feedforward networks (also referred to as subnetworks). SNs have excellent performance when utilized in tasks that measure the similarity between two data inputs [1]. From its initial application to handwritten signature verification [2], SNs have been used in several Machine Learning (ML) fields, such as face recognition [3], semantic similarity analysis [4], and object tracking [5]. One of the most popular ML applications of SNs is image discrimination, i.e., to determine whether two images belong to the same category. Different from a traditional NN, an SN for classification purposes only needs to know whether the two inputs belong to the same category, so it does not require explicit labels. Therefore, SN can be treated as semi-supervised learning and may not achieve a classification accuracy as high as a traditional NN; however, SNs usually achieve superior performance in tasks with a large number of classes and one-shot learning [6].

The hardware implementation of an ANN can be affected by several types of errors. Transient faults (often induced by radiation phenomena, for example) cause soft errors that appear as random bit flips, especially in memory chips; a soft error may lead to data corruption and eventually system failure [8]. If errors occur in the inference process of an ANN, they can change the trained model, or the stored weight values; therefore, errors may have an impact on the final result (e.g., the predicted category in the classification tasks). To ensure the resilience of an ANN in the presence of errors, an error-tolerant design is therefore needed. The error tolerance of ANNs for reliable operations has been extensively studied, especially when they are used in safety-critical applications [7]. For example, schemes based on error correction codes (ECCs) have been proposed [9]-[11] that can be applied for reliable storage of ANN parameters; however, they incur in a 10% to 20% redundant memory overhead, while introducing additional delay and power consumption [12]. Redundancy on neurons or computational units has also been proposed [13], [14], but an increase in hardware and a modification to the model are usually required. As a large memory is needed to store the ANN parameters, its protection has stringent requirements in terms of memory overhead; especially for applications on constrained platforms, such as portable systems.

An SN presents a unique challenge for error tolerance, because weight matrices are shared between the subnetworks, so memories (storing the weights) are very important for the reliable operation of this type of ANNs. Especially in the case when weights are stored in the memory as floating-point (FP) numbers, even a single bit flip on the exponent part of the FP number can result in a substantial change in weight value, thereby propagating through the SN with catastrophic effects.

Manuscript received November 4, 2021, revised April 24, 2022. The research was supported by the Spanish Agencia Estatal de Investigación under Grant PID2019-104207RB-I00 and Grant RED2018-102585-T, by the Department of Research and Innovation of Madrid Regional Authority under Grant Y2018/TCS-5046, and by NSF under Grant CCF-1953961 and Grant 1812467. (*Corresponding author: Shanshan Liu*).

Ziheng Wang, Farzad Niknia and Fabrizio Lombardi are with Department of Electrical and Computer Engineering, Northeastern University, MA 02115, USA.

Shanshan Liu is with Klipsch School of Electrical and Computer Engineering, New Mexico State University, NM 88001, USA (email: ssliu@nmsu.edu).

Pedro Reviriego is with Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid, Madrid 28911, Spain.

Paolo Montuschi is with Dipartimento di Automatica e Informatica, Politecnico di Torino, Torina 10129, Italy.

Fig. 3. Inference process of the error-tolerant SN with weight filter and code.

TABLE I
DIFFERENT ERROR INJECTION SCENARIOS

Scenario		# flipped bits	# affected weights
Simple case		1	1
General case	**Single-bit errors**	1	Up to10
	Multi-bit errors	Up to10	1

multiple data bits in a single memory word, or single data bits in multiple adjacent words [25]. Therefore, two scenarios are defined as general cases. 1) Single-bit errors: multiple errors occur, but each incorrect weight/memory word has only one bit flipped. 2) Multi-bit errors: multiple errors occur but they affect the same weight/memory word. The different error scenarios considered in this paper are summarized in Table I.

To achieve error tolerance of ANNs for inference, redundancy on neuron computation has also been proposed [13], [14]; however, the model must be changed and such a scheme may incur in a significant hardware overhead. An alternative solution is to utilize ECCs, as widely used for memory protection. By storing a few parity bits (that are calculated using the data being protected in each memory word), errors in the word can be detected or corrected. For example, the widely used Single Error Correction (SEC) code requires r parity bits by satisfying $2^r \geq d + r + 1$ to protect a d-bit data [31]. Even though ECC has a high flexibility and does not need to modify the NN, the introduced memory redundancy may not be acceptable in some hardware-constrained platforms; moreover, ECCs can only deal with a limited number of errors. Therefore, more powerful yet efficient error-tolerant techniques need to be investigated for a memory of an ANN used in inference, especially for SNs that present a unique challenge for memory errors.

C. Error Injection Experiments

Prior to introducing the proposed error-tolerant approaches, a series of experiments have been pursued to assess the impact of soft errors on the inference performance of an SN for five widely used image datasets given in Table II; both MLP and CNN are considered as subnetworks (their configurations are described in the next sections). Error injection is performed using the error models described in Section II.B for the simple and general cases; bit-flip errors are randomly injected on the weight matrices in each case. The experiments are repeated by 10^5 trials to obtain the average values, from which several important conclusions can be drawn:

1) Due to the overprovision of neurons and the inference threshold, an SN has some inherent error tolerance [18] because soft errors lead to close to zero predicted changes in most cases (97.66% in 10^5 trials). However, in the remaining cases, errors have a very high impact; an average of 37.14% of the predictions are changed.
2) All these changed predictions are caused by significant changes (usually larger than $\pm 10e^{30}$) in the weight value. By Eq. (1), bit flips on the sign or exponent bits are more likely to lead to significant changes. This has been confirmed by error injection; the results show that 99.42% of them are caused by errors on the exponent bits, and 0.58% on the sign bit.

From the above observations, the primary objective of error tolerance in SNs is to protect the sign and exponent bits, i.e. changes on the mantissa bits have a negligible impact on the final prediction. Therefore, this paper proposes error-tolerant approaches from two perspectives: reducing the impact of significant changes by applying a weight filter and correcting the erroneous bits by a code scheme.

TABLE II
DETAILED INFORMATION OF DATASETS AND INFERENCE ACCURACY WITH SN IN ERROR-FREE CASE

Description of Datasets							Accuracy (MLP)	Accuracy (CNN)
Name	**Category**	**#Classes**	**Size of the training set**	**Size of the test set**	**Size of image**	**#Channels of image**		
MNIST [26]	Handwritten Digits	10	60,000	10,000	28×28	1	97.174%	99.613%
Fashion-MNIST [27]	Clothing and Accessories	10	60,000	10,000	28×28	1	91.563%	93.639%
CIFAR-10 [28]	Animals and Vehicles	10	50,000	10,000	32×32	3	75.704%	79.524%
SVHN [29]	Street View House Numbers	10	73,257	26,032	32×32	3	81.867%	92.883%
STL-10 [30]	Animals and Vehicles	10	5,000	8,000	96×96	3	68.421%	74.690%

D. Proposed SN

The proposed error-tolerant SN is based on utilizing two schemes: a weight filter and a code, either by themselves or combined. It will be shown in subsequent sections that the SN can be protected by those two techniques with a very low accuracy loss even in extreme error cases. Error tolerance in this paper focuses on the memory storing the model (i.e., the weights) of the SN, so as applicable to all NNs, training and inference must be considered: 1) The training process of the proposed scheme is the same as in a traditional SN; 2) The significant difference occurs for inference: a weight filter and a code are utilized to check the stored weight matrices before the inference process starts. In the combined scheme, the weight is first checked and corrected by the code, and then further modified by the weight filter. The inference process then starts with such corrected weights. The inference process of the proposed error-tolerant SN is illustrated in Figure 3.

Both these schemes aim at reducing the redundancy overhead for attaining error tolerance in the SN; different from a traditional approach with ECCs, the proposed design of an error-tolerant SN requires zero memory overhead. A detailed comparison is provided in Section VIII.

III. Proposed Weight Filter

As observed in the error injection experiments, the filter is applied to reduce the significant changes in the weight value. Similar schemes for restricting the large values in the weights have been studied [19]-[22]. All these works rely on the same observation that outliers with large absolute values in weights (caused by faults/errors) are the primary causes of performance degradation of NNs. However, the error models considered in these works are totally different from the one analyzed in this paper, i.e. they target stuck-at faults during the training process, while this paper considers soft error(s) during the inference process. Moreover, these existing works did not provide a theoretical analysis or justify the bound selection of the filters, so their applicability in the scenario considered in this paper cannot be further investigated/evaluated.

The design of the proposed weight filter relies on the distribution of the weight values. The statistical maximum and minimum values of the weights in the error-free SNs for different datasets are provided in Table III. The weight values are concentrated in a small range (bounded by the maximum and minimum values). Therefore, when errors due to bit-flips affect the exponent bits, there is a high probability to generate outliers that exceed the range in the error-free case. Such outliers have been shown in the error injection experiments to cause changes in the prediction; hence, the proposed filter can effectively reduce the effect of soft errors in an SN. Note that even though the specific range of weight values may vary for different SNs and datasets (as shown in Table III), it only affects the bound selection of the filter that is described in subsequent subsections, but not the feasibility of the filter.

A. Implementation

Consider the characteristics of the weight distribution; the filter can be mathematically represented as a multiplication with a rectangular function that constrains the weight values within a reasonable range. Such function is defined as:

TABLE III
Statistical Maximum and Minimum Values of Weight Distributions

Dataset	MLP		CNN	
	Maximum	Minimum	Maximum	Minimum
MNIST	0.3209	-0.2725	0.7454	-0.6915
Fashion-MNIST	0.3764	-0.2566	0.9197	-0.6118
CIFAR-10	0.1392	-0.1839	0.2268	-0.2648
SVHN	0.2788	-0.2489	0.5327	-0.5830
STL-10	0.1932	-0.2245	0.6192	-0.5882

$$weight\_modified = weight \times \mathrm{rect}(weight) \quad (2)$$

$$\mathrm{rect}(t) = \begin{cases} 1, & \text{if } t \geq \tau_{min} \text{ or } t \leq \tau_{max} \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

where τ_{min} and τ_{max} determine the lower and upper bounds of the filter, and $\tau_{min} < \tau_{max}$. As this filter is applied to all weight matrices, the bound of the filter must satisfy the constraints:

$$\begin{cases} \tau_{max} \geq w_{max} \\ \tau_{min} \leq w_{min} \end{cases} \quad (4)$$

This is required because only the weights that certainly contain errors, need to be filtered; otherwise, the weights without errors may also be changed, so hindering the objective of retaining high accuracy while reducing the impact of memory errors.

The bounds of the filter are recorded when the model is saved in memory. Prior to the inference process, the weight matrices are checked by comparing them with the upper and lower bounds. After the outliers are limited by the filter, the inference process continues with the modified weights (as illustrated in Figure 3). Next, the specific choices and related analyses of the upper and lower bounds of the filter are discussed. Due to the complexity of NN propagation, a simple model is first introduced for the theoretical analysis, and then the general cases are discussed.

B. Upper and Lower Bounds in the Simple Case

A simplified error injection model is considered to analyze the weight filter in the proposed SN. The following assumptions are applicable to the so-called "simple case" of an SN.

1) Only one erroneous bit is present in the weight matrices.
2) The output layer of each branch network has only one neuron.
3) The error appears only in the weight of the last layer (i.e., the output layer in the MLP version, or the last fully-connected layer in the CNN version).

As an example, consider one of the branch networks, because these networks share the same weights and errors are the same for their weight matrices. Define the inputs of the last layer of subnetwork 1 as $x_1 = (x_{1,1}, \cdots, x_{1,n})'$, and the weight matrix as $w_1 = (w_{1,1}, \cdots, w_{1,n})'$, where n is the number of neurons prior to the output layer. The weight with a memory error $w_i + e$ has a range from $\pm 1.18 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$ as per a single bit-flipping error in a single-precision floating-point number.

Assume the error e affects $w_{1,i}$, $1 \leq i \leq n$. The output of the last layer can be represented by Eq. (5); only the weight is considered, because the bias can be treated as part of the weight if the input is extended by an additional dimension (i.e., $w_1 = (w_{1,1}, \cdots, w_{1,n}, b)'$ and $x_1 = (x_{1,1}, \cdots, x_{1,n}, 1)'$).

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2022.3186628

$$y_1 = \sum_{j=1}^{n} x_{1,j}\, w_{1,j} + x_{1,i} e \tag{5}$$

The output of the other subnetwork, y_2 can be similarly represented. When the output layer has only one neuron, the calculation of the Euclidean distance is rather simple; the distance D_0 with no error and the distance D with one bit of error satisfy the following relationship.

$$\begin{aligned} D &= \sqrt{(y_1 - y_2)^2} \\ &= \sum_{j=1}^{n} x_{1,j}\, w_{1,j} + x_{1,i} e - \sum_{j=1}^{n} x_{2,j}\, w_{2,j} - x_{2,i} e \\ &= D_0 + (x_{1,i} - x_{2,i}) e \end{aligned} \tag{6}$$

Assume the value of D_0 is in the range $[0, D_{max}]$, and the threshold is T, $0 < T < D_{max}$. As per the classification results of the network with no error, the following two cases are possible:

Case 1: If $D_0 \in [0, T]$, the original prediction is positive (i.e., the two inputs are judged by the SN to be of the same category). To reduce the effect of errors, i.e., to keep the prediction unchanged, $D_0 + (x_{1,i} - x_{2,i}) e \in [0, T]$. If $(x_{1,i} - x_{2,i}) e \le 0$, the prediction does not change; if $(x_{1,i} - x_{2,i}) e > 0$, the filter reduces $(x_{1,i} - x_{2,i}) e$ to reach a higher probability for the prediction not to change.

Case 2: If $D_0 \in [T, D_{max}]$, the original prediction is negative (i.e., the two inputs are judged by the SN to be of different categories). Similarly, $D_0 + (x_{1,i} - x_{2,i}) e \in [T, D_{max}]$. If $(x_{1,i} - x_{2,i}) e \ge 0$, clearly the prediction does not change. if $(x_{1,i} - x_{2,i}) e < 0$, the filter increases $(x_{1,i} - x_{2,i}) e$ to reach a higher probability for the prediction not to change.

By combining the above two cases, the selection of the upper and lower bounds of the filter is equivalent to minimize $|(x_{1,i} - x_{2,i}) e|$. Moreover, as the filter is applied to the weight matrices and does not change the inputs, the above problem is equivalent to minimize $|e|$. The filter accomplishes this indirectly by limiting the value of $|w_i + e|$ as:

$$w_i + e = \begin{cases} \tau_{max}, & if\ w_i + e > \tau_{max} \\ \tau_{min}, & if\ w_i + e < \tau_{min} \\ w_i + e, & otherwise \end{cases} \tag{7}$$

When considering (4), the selection of the bounds is equivalent to solving the problem:

$$\begin{cases} \text{minimize } \tau_{max}, & subject\ to\ \tau_{max} \ge w_{max} \\ \text{maximize } \tau_{min}, & subject\ to\ \tau_{min} \le w_{min} \end{cases} \tag{8}$$

The optimal feasible solution for the filter's upper and lower bounds is given by:

$$\begin{cases} \tau_{max} = w_{max} \\ \tau_{min} = w_{min} \end{cases} \tag{9}$$

This shows that the filter with bounds given by the statistical maximum and minimum of the weight values (as presented in Table III) has the largest probability to maintain the original predictions in the simple case.

C. Theoretical Performance Analysis

By observing the distributions of the weights and the distances in the error-free case, the following relationships are satisfied for all five datasets (described in Table II) considered in this paper:

TABLE IV
ESTIMATED PERCENTAGE OF PROTECTED SAMPLES IN DIFFERENT DATASETS

Dataset	Protected samples (%)
MNIST	82.98%
Fashion MNIST	73.88%
CIFAR-10	78.32%
SVHN	78.59%
STL-10	80.35%

$$\begin{cases} \left|(x_{1,i} - x_{2,i}) w_{min}\right| < D_{max} - T \\ \left|(x_{1,i} - x_{2,i}) w_{max}\right| < T \end{cases} \tag{10}$$

Therefore, by applying the filter, a "safe range" is generated to guarantee that most of the predictions remain unchanged. For example, if there is a constant ϵ such that $(x_{1,i} - x_{2,i}) e < \epsilon_1$, for the samples with an original output distance $D_0 \in [0, T - \epsilon]$, their output distance in the presence of errors always satisfies $D = D_0 + (x_{1,i} - x_{2,i}) e < T$. The derivation is similar if the sample's prediction is negative with $(x_{1,i} - x_{2,i}) e > \epsilon_2$. This means that the positive predictions with an original distance $D_0 \in [0, T - \epsilon_1]$, or the negative predictions with an original distance $D_0 \in [T + \epsilon_2, D_{max}]$ do not change.

Assume that an error does change the sign of the weight; note that the probability of a random bit flipping on the sign bit of a single-precision floating-point number is very small (1/32), so this is an extreme scenario. Then the following inequalities are satisfied by applying a filter according to Eq. (9).

$$\begin{cases} e \le w_i + e \le w_{max}, & \text{if } w_i \ge 0 \\ e > w_i + e \ge w_{min}, & \text{if } w_i < 0 \end{cases} \tag{11}$$

Define $\epsilon = (x_{1,i} - x_{2,i}) w_{max}$, such a "safe range" is generated; therefore, by the distribution of $(x_{1,i} - x_{2,i}) w_{max}$ and $(x_{1,i} - x_{2,i}) w_{min}$ of each dataset, the estimated percentage of samples protected by the filter can be calculated and for different datasets, it is shown in Table IV. As per Eq. (11), this analysis results in an underestimate of the performance of the filter; so, it can be theoretically proven that the filter is capable to reduce the changed predictions caused by memory errors.

D. General cases

The optimality of the bound selection has been established in the simple case; in the general case, memory errors in an SN may not satisfy the assumptions outlined previously in Section III.B, and a more detailed analysis is required. The proposed weight filter works also in the general case, because its feasibility only relies on the condition that outliers (so with large absolute values in the weights) lead to degradation. However, it has not been possible to prove that the specific bound selection is optimal as in the simple case (also, the theoretical performance analysis is intractable). Therefore, this paper provides a discussion of a heuristic bound selection for the general cases.

If assumptions 1) or 2) are not satisfied, Eq. (6) can be changed by substituting the error terms with the sum given by

$$D = D_0 + \sum_{i=1}^{k} (x_{1,i} - x_{2,i}) e_i \tag{12}$$

where k is the number of soft errors, or the number of output dimensions. In this case, the problem is equivalent to minimize $|\sum e_i|$. The optimal solution of Eq. (9) only holds when all errors have the same sign, which is not necessarily valid in

Fig. 4. Data represented in IEEE standard 754 floating-point format protected by the proposed code scheme.

practice; hence, the bounds are not always optimal (depending on the signs of the errors), but it is a feasible solution for all cases.

When assumption 3) is not satisfied, the analysis becomes rather complicated. Even if the filter minimizes the changes in the erroneous layer, the final output cannot be predicted; this also happens to the convolution layers of the CNN version. It occurs because errors cannot be represented as an independent term (as in Eq. (6)) after layer propagation. In this case, the optimality of the bounds is highly dependent on the network and the datasets, so it is difficult to establish a deterministic mathematical model.

Even though optimality does not always hold, the bound selection in Eq. (9) can be still used. Considering the diverse datasets, the bounds set by the range of the weights provide good flexibility; also, it is generally applicable for all scenarios (optimal in the simple case; feasible and sometimes optimal in the more general cases). As per the observation that outliers are usually extremely large, a small deviation to the optimal bounds brings trivial effects; the performance of the filter with the suggested bounds is evaluated by simulations in subsequent sections.

IV. Proposed Code

A code scheme is proposed to deal with single-bit errors in this section. Since outliers due to errors on the sign and exponent bits of weights cause the most crucial changes affecting predictions, the protection of only these bits can greatly reduce the redundancy overhead, while still retaining satisfactory performance. Therefore, in the proposed code scheme, an SEC is employed to cover only the sign and exponent bits (i.e., 9 bits in total) of each weight; this reduces the required number of parity bits from 6 to 4 compared to a traditional SEC that covers all 32 floating-point bits [31]. Moreover, the proposed scheme stores the parity bits on the 4 least significant bits (LSBs) of the mantissa of each weight (Figure 4), hence no memory redundancy/overhead is finally encountered. Even though a very small deviation is introduced by replacing the original 4 LSBs with the parity bits, it leads to negligible changes in values and operations of ANNs [32]; this is also applicable to an SN as established in the error injection experiments.

Overall, this scheme is expected to have a significant performance improvement either by itself, or when combined with the filter (when there is more than one erroneous bit on the significant bits, the provided code is not suitable, hence the need for the weight filter).

V. Simulated Error Tolerance of SN with MLPs

In this section, the performance of the proposed schemes (filter, code, and the combined method) in an SN with different datasets is assessed for both the simple and the general cases. Multi-layer perceptrons (MLPs) are used as branch networks and each is designed with 5 linear layers (with size 784-512-512-512-2); ReLU is selected as the activation function between layers. The number of layers, the number of neurons in each layer, and the hyperparameters (such as the learning rate) are experimentally determined for best performance over all datasets. Moreover, contrastive loss [33] is used in the training of the SN. As an SN requires a pair of images as inputs for the two branch networks, datasets are preprocessed to balanced positive and negative pairs, so indicating that the two inputs belong to the same or different categories.

The SN is trained with the preprocessed datasets and the classification accuracy of each dataset in the error-free case is recorded and shown in Table II; in the simulations, the SNs for all datasets are equally trained for 50 epochs. Although the models may not reach the top accuracy of each dataset, this has a negligible effect on testing the weight filter in the presence of error(s).

Errors for different cases (as detailed in Table I) are injected into the SN during the inference process. The location of each error (layer, weight index, and bit position) is randomly selected. Two metrics are used to evaluate the effects of the injected errors and the proposed schemes: the loss classification accuracy and the number of changed predictions (all results are averaged over 10000 runs). Since the models cannot achieve 100% accuracy in the error-free case, there will be differences between these two metrics. The accuracy loss evaluates the classification performance of the network, while the number of changed predictions evaluates the sensitivity of the model to errors at a higher system level.

TABLE V
Performance of the Proposed Methods in the Simple Case

Dataset	Accuracy loss			# of changed predictions		
	Un-protected	Filter	Code	Un-protected	Filter	Code
MNIST	0.030%	0.007%	<0.001%	2.582	0.505	0.028
Fashion MNIST	0.013%	0.002%	<0.001%	3.287	1.187	0.035
CIFAR-10	0.003%	<0.001%	<0.001%	1.334	0.336	0.044
SVHN	0.002%	<0.001%	<0.001%	1.950	0.409	0.018
STL-10	0.017%	0.002%	<0.001%	5.744	1.081	0.013

A. Performance in the Simple Case

In the simple case, only a single bit error is injected into the weight matrices for the output layer. The results prior to and after using the proposed filter and code, are shown in Table V; note that there is no need to employ the combined method in this case.

Filter scheme: Table V shows that the filter achieves improvement in both accuracy loss and the number of changed predictions. Moreover, in Section III, it has been theoretically proved that the filter can protect part of the samples and reduce the impact of memory errors on their predictions in the simple case. Although this theoretical calculation is only an estimate result (given in Table IV), its difference compared with the simulated number of changed predictions (given in Table V) is only 3.44% on average for all datasets, so verifying the analysis of the filter in the simple case.

(a) (b) (c) (d)

Fig. 5. loss of each scheme under single-bit errors: (a) unprotected; (b) with filter; (c) with code; (d) with combined method.

(a) (b) (c) (d)

Fig. 6. Number of changed predictions of each scheme under single-bit errors: (a) unprotected; (b) with filter; (c) with code; (d) with combined method.

Code scheme: In this scheme, the single-bit error occurred in the memory can be fully corrected; therefore, the performance of the code is superior to the filter. Since the original 4 LSBs of each weight have been replaced by the parity bits, an accuracy loss and/or the number of changed predictions are still incurred for some datasets (as expected); however, such degradations are extremely low compared to the unprotected scheme and filter scheme (shown in Table V).

Overall, in the simple case, the code scheme is more attractive due to its higher error-tolerant performance with no memory overhead.

B. Performance in the General Case – Single-bit Errors

In this subsection, the performance of the proposed filter, code, and the combined method is assessed in the presence of up to 10 single-bit errors. The accuracy loss and the number of changed predictions are provided in Figures 5 and 6. The results show that as expected, the values under both metrics increase with the number of errors (as flipped bits in the unprotected scheme), and a significant reduction is achieved using the proposed methods.

Filter scheme: When comparing the results without and with the filter, the accuracy loss and the number of changed predictions are in general reduced by more than 90%; this confirms that the filter significantly reduces the effect of single-bit errors. In practical applications, the accuracy of the network usually decreases sharply with an increasing number of errors, so the presence of a filter enables to significantly reduce the impact of errors.

(a) (b) (c) (d)

Fig. 7. loss of each scheme under multi-bit errors: (a) unprotected; (b) with filter; (c) with code; (d) with combined method.

(a) (b) (c) (d)

Fig. 8. Number of changed predictions of each scheme under multi-bit errors: (a) unprotected; (b) with filter; (c) with code; (d) with combined method.

Code scheme: The code achieves even better performance for single-bit error correction; it shows very low accuracy losses (< 0.001%) and the average number of changed results is generally less than 0.3. Different from the filter, the code scheme ensures that the significant bits are error-free, so it is nearly insensitive to the number of errors. This feature enables a substantial improvement in performance under a large number of single-bit errors.

Combined scheme: The performance of the combined method is very similar to employing only the code (i.e., Figure 5 (c) and 6 (c)), thus proving that the combined method fully retains the excellent single error correction capability of the code. This result is expected, because the weights corrected by the code fall into the allowed range of the filter; hence the filter is not activated for most of the single error cases and the significant bits of the weights are still fully protected by the code.

C. Performance in the General Case – Multi-bit Errors

In this subsection, the proposed filter, coding and the combined method are assessed in the presence of up to 10 multi-bit errors; the accuracy loss and the number of changed predictions are provided in Figures 7 and 8.

Filter scheme: The results show that the filter maintains its good performance also under multiple error bits. It performs slightly better than the case of single-bit errors (Figures 5 (b) and 6 (b)). This is reasonable, because multi-bit errors are more likely to generate outliers with a large absolute value, that can be effectively eliminated. Moreover, with the protection of the filter, the accuracy of the network does not further decrease when more flipped bits occur in a single weight. This confirms that the proposed filter is suitable for protecting an SN with multiple erroneous bits in the same weight.

Code scheme: The code scheme cannot protect against multi-bit errors due to its nature (i.e., the SEC code); this is confirmed by the results showing no improvements (except when only one error occurs).

Combined scheme: The combined method has satisfactory performance in the case of multi-bits errors. The result is very close to using the filter, and the small difference is due to the changed LSBs (replaced by parity bits in the code scheme).

Overall, the experiments for the general cases have shown that the combined method retains the advantages of both the filter and the code, so protecting the SN in both cases of single-bit and multi-bit errors.

VI. Simulated Error Tolerance of SN with CNNs

In previous sections, the performance of the proposed error-tolerant schemes has been evaluated for an SN with MLPs as subnetworks. This section provides additional simulation results by considering an SN with CNNs. Each of these subnetworks consists of three convolution layers (with channels and kernel size of 64×7×7-128×5×5-256×5×5) and two fully-connected layers (with size 2304-2); ReLU is used as the activation function. The number of layers, the number of neurons in each layer, and hyperparameters (such as the learning rate) are experimentally determined for best performance over all datasets. Moreover, contrastive loss [33] is used in the training of the SN; dataset preprocessing and other configurations in the SN are the same as for the MLP-based version.

The performance of the proposed weight filter, coding scheme, and the combined method is presented in Tables VI and VII. Due to space limitations in the manuscript, only the results of 10-bit flips for each error type in the general cases (i.e., single-bit and multiple-bits errors) are provided. The results have been averaged over 10000 repeated trials.

The results show that the filter protects against both single-bit and multiple-bits errors; it performs better in the case of multiple-bit errors. The code provides a nearly complete protection for the SN in the case of single-bit errors, but it does not work well under multi-bit errors. Also in the CNN-based version, the combined method has the advantage of the two proposed schemes. Performance of the CNN-based version is very similar to the MLP-based version, so showing the proposed schemes can also work effectively with CNNs as subnetworks.

TABLE VI
PERFORMANCE OF THE PROPOSED ERROR-TOLERANT SCHEMES WITH CNN IMPLEMENTATION EVALUATED BY ACCURACY LOSS (%)

Dataset	Unprotected		Filter		Code		Combined method	
	Single-bit errors	Multi-bits errors	Single-bit errors	Multi-bits errors	Single-bit errors	Multi-bits errors	Single-bit errors	Multi-bits errors
MNIST	1.118%	2.767%	0.109%	0.070%	0.004%	2.754%	0.005%	0.072%
Fashion MNIST	0.776%	1.533%	0.199%	0.082%	0.007%	1.467%	0.006%	0.079%
CIFAR-10	1.047%	1.958%	0.071%	0.066%	0.004%	1.937%	0.004%	0.062%
SVHN	0.681%	1.886%	0.062%	0.042%	0.006%	1.786%	0.007%	0.044%
STL-10	0.657%	2.255%	0.112%	0.072%	0.009%	2.252%	0.010%	0.065%

TABLE VII
PERFORMANCE OF THE PROPOSED ERROR-TOLERANT SCHEMES WITH CNN IMPLEMENTATION EVALUATED BY CHANGED PREDICTIONS

Dataset	Unprotected		Filter		Code		Combined method	
	Single-bit errors	Multi-bits errors	Single-bit errors	Multi-bits errors	Single-bit errors	Multi-bits errors	Single-bit errors	Multi-bits errors
MNIST	137.46	365.53	16.65	13.25	1.02	356.98	1.13	13.10
Fashion MNIST	109.17	291.24	33.59	23.73	0.63	292.35	0.75	22.81
CIFAR-10	127.91	519.41	25.18	21.16	2.27	501.29	2.16	21.75
SVHN	183.51	703.77	28.78	22.90	1.28	687.78	1.31	20.49
STL-10	263.51	543.81	37.95	19.87	0.98	541.96	0.96	19.55

(a) (b)

Fig. 9. An MLP: (a) its structure; (b) computation of a hidden/output neuron.

VII. Hardware Implementation of SN with MLP

In this section, a high-performance ASIC design of an SN is pursued for use in hardware-constrained platforms (e.g., low-power/high-frequency IoT or mobile systems). Since a CNN is computationally intensive, implementations often use FPGAs [34], GPUs [35], or supercomputers [36]; moreover, the operational frequency for FPGA-based implementations is in the order of few hundreds megahertz, and considerable power dissipation (like tens of watts) is encountered for CPU/GPU-based implementations [37], so making CNNs unsuitable as branch networks in an SN (so doubling the size of a CNN) for hardware-constrained platforms. Therefore, only an SN with MLPs as subnetworks is considered next for ASIC design; an efficient CNN-based SN implementation using emerging computing paradigms (so not with FP numbers) will be pursued as future work.

As depicted in Figure 9 (a), an MLP has an input layer, at least one hidden layer and an output layer. Generally, the number of neurons in the input layer is equal to the valid feature's dimension in the target dataset; a weight is applied when mapping a neuron's value to the next layer. Therefore, each neuron must be multiplied by its relevant weight to be transferred to the next layer.

Consider Figure 9 (b); when calculating the value of neuron n_{i+1} in the i+1th layer, all m neuron values in layer i are initially multiplied by the corresponding weights w of neuron n_{i+1}. Then these values are accumulated and finally, the activation function Φ is applied. This calculation is given in Eq. (13).

$$n_{i+1} = \Phi\left(\sum_{j=1}^{m} w_{j,i+1}^{i} \cdot n_j + b^i\right) \quad (13)$$

Therefore, when using the FP numbers (as numerically defined in Eq. (14), where h is the hidden bit), a Multiply-Accumulate (MAC) unit consists of an FP adder and an FP multiplier as required to implement a neuron of an MLP.

$$(-1)^{sign} \times 2^{(exponent-127)} \times (h.mantissa) \quad (14)$$

Next, the FP units that are used for computation are presented. The 2-input FP adder and multiplier have been implemented using a 4 states finite state machine in which each state is processed in a clock cycle; the output is updated every 4 clock cycles. Algorithms 1 and 2 (given in the supplemental material) illustrate in more detail the process of addition and multiplication using these designs; both the FP adder and multiplier are designed behaviorally using Verilog-HDL. For the distance computation at the final stage of the SN, a square root circuit design is required; the low-cost design (and related method) proposed in [38] is utilized in this paper. This circuit is also described in the supplemental material.

Fig. 10. Serial MLP implementation (Address sizes are for MNIST & FMNIST).

Different schemes can be investigated for the MLP and thus SN implementation: 1) Serial 2) Parallel 3) Hybrid. The parallel implementation is presented in the supplemental material, because due to its very large complexity, this design is not pursued because the hardware is unfeasible. In the next subsections, the remaining two implementations are discussed in more detail.

A. Serial Implementation of MLP

In a serial implementation of an MLP, all calculations are performed by utilizing a single MAC [39] (as part of the structure shown in Figure 11); this design receives two inputs and multiplies them in 4 clock cycles in a pipelined mode. Then, it adds the product with the results from the previous step using the feedback input in the FP adder. Also, in this implementation when the start signal is low, all internal registers of the FP units are reset, otherwise the multiplier starts the calculation.

Assume that the i^{th} layer has m neurons; for each neuron in the i+1th layer, m+1 entries (so the number of neurons in the i^{th} layer plus a bias value) are present, and thus m+1 pairs of inputs. After calculating the value of each neuron, the MAC is reset and the calculation for the next neuron can be started. Therefore, a serial implementation computes each neuron value for the hidden and output layers serially; this feature significantly decreases the area, but it increases the latency. Figure 10 shows a schematic diagram of this implementation; an SRAM is required to save the result of each neuron, because this value is needed for the calculations of the neurons in the deeper layers. Also, a portion of the SRAM is used to save the weights related to each neuron; the control unit is responsible for reading/writing from the SRAM and set/reset of the neuron [40].

B. Hybrid Implementation of MLP

This implementation is based on a combination of parallel and serial implementations; it uses the serial neuron hardware (i.e., a MAC). Consider the fully parallel design in Figure 11 (more details about this design can be found in the supplemental material file); the hybrid design consists of increasing the level of parallelization by adding more MAC units to the design. So, instead of calculating each neuron separately, several neurons are calculated at the same time, but still in a serial mode, i.e., as each neuron processes only a pair of inputs in each step, the control unit allocates all possible pairs to the neuron during the computational phase.

Therefore, the final values of neurons are saved in the SRAM

Fig. 11. Parallel MLP implementation.

Fig. 12. Hybrid MLP implementation (Address sizes are for MNIST & FMNIST).

and used for the next layers (because they cannot be processed at the same time due to the limited number of MACs). In this paper, 16 MAC units are used for the different datasets considered; however, the SRAMs must be implemented such that several banks are provided to either generate 16 outputs at the same time, or save 16 inputs at the same time. In this case the performance of the SRAM memory plays an important role in the proposed design.

Figure 12 shows the proposed design for the hybrid implementation; in this design, only a single array of MACs is utilized. Therefore, for each layer, the control unit sets the proper inputs for the MACs (weights and input data) and when the computation of that layer is completed, the results are saved in the related SRAM, because they are needed as inputs of the next layer (so different from a fully parallel implementation). For computation of the next layers, the current array of MACs can be used again. As memories, SRAM_W stores the weights and bias values, while SRAM_N saves the neurons values.

C. *Proposed SN Implementation*

To implement the SN, the MLP design is utilized; To be more specific, a single MLP unit (Serial or Hybrid) is instantiated, and all calculations of the SN are calculated in 2 iterations (1 iteration for each pair of networks). Also, as the used weights for both subnetworks are the same, then the SRAM_W and SRAM_N units in the MLP design are sufficient. The control unit controls the iterations. Additionally, the input image must be multiplied with its related weights several times (the hardware design has 1 or 16 MAC units and there are 512 neurons in the first hidden layer, the image must be accessed 512 or 32 times for serial and hybrid designs respectively), then a small SRAM unit is used to save a pair of images for next time (SRAM_I). Figure 13 shows the proposed SN design. Based on Table VIII, the power consumption approximately doubles (there are 2 iterations) and the area increases due to the control unit is marginal. This allows for ease in training and testing because both subnetworks use the same weights and bias values; latency however may increase. These aspects will be assessed in the next section.

Fig. 13. SN Design (Based on serial MLP design for MNIST & FMNIST).

VIII. Evaluation Results

All hardware implementations have been designed/simulated using Cadence Genus Synthesis Solution; the optimization effort for area, power and delay has been set to high for the tool to automatically consider the tradeoffs between them and the pre-set constraints to reach the best result. Moreover, we have used a 32nm technology file at 25°C and TT corner. Table VIII shows the synthesis results for the different datasets for the MLP and SN using serial and hybrid implementations based on a 32nm library (with and without error-tolerant schemes). In Table VIII, the serial design incurs in the least area and power dissipation, but at a higher number of clock cycles. So, the serial implementation is the best design candidate for low power applications; however, training can be rather time-consuming. The proposed ASIC implementations operate in a pipeline mode at 681.2 MHz frequency, while the total delay during classification is several milliseconds.

Memory is needed for the serial and hybrid designs to save the values of the deeper neurons because these designs do not use all of them at the same time. Although the memory size for these two designs are the same (Table VIII), their control units are rather different. In the serial design, the data is read/written serially during each cycle, while for the hybrid design the data is written in the hybrid mode at different memory banks (16 data banks); during the read cycle, the data is read in the sequence that they have saved, so serially as multiplication is executed between the value of each neuron in the previous layer and its related weight for a neuron in the current layer. Note that errors in these memories (that store the intermediate calculation results in both the serial and hybrid designs) are not considered in this paper, because their size is comparatively small and traditional error-tolerant schemes can be utilized to protect them against errors (if needed). Also, the reason for the difference between memory size in MLP design and SN design is that in a SN design, another memory has been added to save the input pair of images.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2022.3186628

TABLE VIII
SYNTHESIS RESULTS OF DIFFERENT DESIGNS

Dataset	Design	Serial Implementation						Hybrid Implementation					
		ASIC				Memory		ASIC				Memory	
		Power (*mW*)	Area (mm^2)	Delay (*ps*)	# Cycles	size (*Mb*)	Area (mm^2)	Power (*mW*)	Area (mm^2)	Delay (*ps*)	# Cycles	size (*Mb*)	Area (mm^2)
MNIST	MLP	1.91	0.0120	1452	3725348	3.547	96.13	21.69	0.1597	1468	239262	3.547	96.42
	SN	4.42	0.0134	1458	7450697	3.553	96.30	43.98	0.1605	1473	478525	3.553	96.59
	SN+fliter	4.88	0.0137	1624				50.54	0.1652	1644			
	SN+code	4.50	0.0135	1586				46.32	0.1621	1628			
	SN+combined	5.26	0.0142	1623	9313357			56.38	0.1706	1644	591777		
	SN+SEC [31]	4.76	0.0138	1731	7450697	4.220	114.36	53.97	0.1666	1773	478525	4.220	114.70
Fashion MNIST	MLP	1.91	0.0120	1452	3725348	3.547	96.13	21.69	0.1597	1468	239262	3.547	96.42
	SN	4.42	0.0134	1458	7450697	3.553	96.30	43.98	0.1605	1473	478525	3.553	96.59
	SN+fliter	4.62	0.0135	1641				46.22	0.1639	1641			
	SN+code	4.50	0.0135	1586				46.32	0.1621	1628			
	SN+combined	4.90	0.0141	1640	9313357			54.84	0.1705	1641	591777		
	SN+SEC [31]	4.76	0.0138	1731	7450697	4.220	114.36	53.97	0.1666	1773	478525	4.220	114.70
CIFAR -10	MLP	2.17	0.0120	1452	4216868	4.015	104.17	24.64	0.1597	1468	271542	4.015	104.47
	SN	5.02	0.0134	1458	8433737	4.023	104.38	49.96	0.1605	1473	543085	4.023	104.67
	SN+fliter	5.34	0.0135	1644				53.48	0.1653	1644			
	SN+code	5.12	0.0135	1586				52.58	0.1621	1628			
	SN+combined	5.6	0.0142	1643	10542157			63.26	0.1715	1644	668577		
	SN+SEC [31]	5.39	0.0138	1731	8433737	4.780	123.95	61.29	0.1666	1773	543085	4.770	124.29
SVHN	MLP	2.17	0.0120	1452	4216868	4.015	104.17	24.64	0.1597	1468	271542	4.015	104.47
	SN	5.02	0.0134	1458	8433737	4.023	104.38	49.96	0.1605	1473	543085	4.023	104.67
	SN+fliter	5.42	0.0137	1622				56.2	0.1639	1641			
	SN+code	5.12	0.0135	1586				52.58	0.1621	1628			
	SN+combined	5.72	0.0141	1620	10542157			65.86	0.1705	1641	668577		
	SN+SEC [31]	5.39	0.0138	1731	8433737	4.780	123.95	61.29	0.1666	1773	543085	4.770	124.29
STL-10	MLP	10.93	0.0120	1452	20989988	20.078	519.24	122.87	0.1598	1468	1352886	20.078	519.54
	SN	25.12	0.0135	1458	41979977	20.086	521.07	249.14	0.1606	1473	2705773	20.086	521.36
	SN+fliter	26.04	0.0137	1638				274.92	0.1651	1633			
	SN+code	25.58	0.0136	1586				262.26	0.1622	1628			
	SN+combined	28.52	0.0142	1638	52485197			302.1	0.1709	1633	3290017		
	SN+SEC [31]	26.81	0.0138	1731	41979977	23.850	618.77	305.68	0.1666	1773	2705773	23.840	619.12

The SNs protected using the traditional SEC code of [31] are also implemented and compared in Table VIII. Compared to the proposed schemes, the SNs with SEC show comparable power and area overhead for the ASIC, but the delay is increased by 5.36% to 8.57%; moreover, the memory size (and thus its area and power) is increased by approximately 18.8% due to the additional cells for storing the parity bits (while only correcting single-bit errors). Since memory accounts for the largest part of the overhead required for the entire implementation, the proposed schemes (that can deal also with multi-bit errors are shown to be superior to the traditional SEC scheme under all evaluation metrics, so making it also more attractive for hardware-constrained platforms.

Finally, the MLP implementation is compared with existing works (based on FPGA and ASIC [41]-[45]) found in the technical literature in terms of hardware performance; the MNIST dataset is taken as an example, but the trend of the other datasets are similar, because the benefit of the proposed design comes from a hardware configuration that is mostly independent of the network configuration/datasets. The synthesis results given in Table IX show that the proposed design achieves the highest frequency, with the least power dissipation even though the existing works use more advanced process technology and have an extremely smaller network.

TABLE IX
COMPARISON OF THE PROPOSED MLP IMPLEMENTATION WITH OTHER MLP DESIGNS FOR MNIST

MLP Implemetation	Technology	Network Topology	Frequency (MHz)	Power (mW)	Arithmetic Units
[41]	FPGA-28 nm	7-6-5	100	120	Fixed-Point
[42]	FPGA-28 nm	784-32-10	100	654	Fixed-Point
[43]	FPGA-28 nm	784-600-600-10	490.8	-	Fixed-Point
[44]	FPGA-16 nm	784-12-10	100	568	Floating-Point
[45]	ASIC-28 nm	784-200-100-10	114.7	54	Floating-Point
Proposed Design	ASIC-32 nm	784-512-512-512-2*	681.2	21.69	Floating-Point

TABLE X
OVERALL PERFORMANCE OF THE PROPOSED ERROR-TOLERANT SCHEMES

Scheme	Decrease in accuracy loss		Decrease in changed predictions	
	Single-bit errors	Multi-bit errors	Single-bit errors	Multi-bit errors
Weight filter	93.542%	97.662%	91.661%	95.288%
Code	99.775%	3.385%	99.545%	2.089%
Combined method	99.782%	97.634%	99.547%	95.315%

IX. CONCLUSION AND FUTURE WORK

This paper has proposed an error-tolerant Siamese Network (SN); two schemes utilizing a weight filter and a code reduce the effect of memory errors (bit flips in the weight matrices) in the inference process. The proposed schemes introduce no memory redundancy compared with the unprotected case, because the filter does not incur in redundancy and the coding scheme replaces a few original LSBs with the required parity bits. Moreover, they can be combined to further improve the error tolerance of the SN. Overall, the performance of the proposed three schemes is summarized in Table X, which is evaluated by the decreased percentage of accuracy loss and number of changed predictions (averaged over all datasets and all error scenarios).

The weight filter has been introduced to prevent outliers and protect the SN in extreme cases in which multi-bit errors occur in a single weight. The theoretical analysis proves that the selection of the filter bounds according to the maximum and minimum values of the weight distribution of the trained model is optimal in the simple case, and feasible in the more general cases. Moreover, the percentage of weights protected by the filter can be calculated in a simple case, so providing an estimate of the filter performance. The code scheme stores the parity bits on the LSBs of the original data, so requires zero memory overhead at the cost of incurring in negligible errors.

Simulation results have shown that the filter performs better in the case of multi-bit errors, while the coding scheme shows better performance under single-bit errors (but it cannot operate correctly for multi-bit errors). When combining both the weight filter and the code, the SN has been shown to have outstanding performance for all error scenarios, so achieving a high memory error tolerance.

The ASIC design of a Multi-Layer Perceptron (MLP) and then an SN (using MLPs as branch networks) with different configurations have been accomplished by using single-precision FP arithmetic units. Synthesis results show that the proposed hardware MLP design provides a higher operating frequency (681.2 MHz) while lower power dissipation compared with other works found in the technical literature, even though these works use more advanced process technologies and have smaller networks.

The proposed methods are also applicable to other types of ANNs when the same behavior of the weight outliers is observed. Future work will investigate i) the efficient hardware design for CNN-based SNs using emerging computing paradigms, and ii) the use of deep NNs as subnetworks of an SN as well as the classification performance and error tolerance of such configuration, because this type of network can also be used in other application scenarios in place of shallow MLPs/CNNs.

REFERENCES

[1] D. Chicco, "Siamese neural networks: An overview," *Artificial Neural Networks,* pp. 73-94, 2021.

[2] J. Bromley, J. W. Bentz, L. Bottou, et al., "Signature verification using a "siamese" time delay neural network," *International Journal of Pattern Recognition and Artificial Intelligence,* vol. 7, pp. 669-688, 1993.

[3] L. Song, D. Gong, Z. Li, et al., "Occlusion robust face recognition based on mask learning with pairwise differential siamese network," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019.

[4] Z. Liang and J. Shen, "Local semantic siamese networks for fast tracking," *IEEE Transactions on Image Processing,* vol. 29, pp. 3351-3364, 2019.

[5] L. Bertinetto, J. Valmadre, J. F. Henriques, et al., "Fully-convolutional siamese networks for object tracking," in *European conference on computer vision*, 2016.

[6] G. Koch, R. Zemel, R. Salakhutdinov, et al., "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, Lille, 2015.

[7] P. Koopman and M. Wagner, "Autonomous vehicle safety: An interdisciplinary challenge," *IEEE Intelligent Transportation Systems Magazine,* vol. 9, no. 1, pp. 90-96, 2017.

[8] N. Kanekawa, E. H. Ibe, T. Suga, et al., "Dependability in electronic systems: mitigation of hardware failures, soft errors, and electro-magnetic disturbances," in *Springer Science & Business Media*, 2010.

[9] J. W. Schwartz and J. K. Wolf, "A systematic (12,8) code for correcting single errors and detecting adjacent errors," in *IEEE Transactions on Computers*, vol. 39, no. 11, pp. 1403-1404, 1990.

[10] S. Pontarelli, P. Reviriego, M. Ottavi and J. A. Maestro, "Low delay single symbol error correction codes based on Reed Solomon codes," in *IEEE Transactions on Computers*, vol. 64, no. 5, pp. 1497-1501, 2015.

[11] S. Liu, P. Reviriego, F. Lombardi, "Detection of limited magnitude errors in emerging multilevel cell memories by one-bit parity (OBP) or two-bit parity (TBP)," in *IEEE Transactions on Emerging Topics in Computing* 2019 (early access).

[12] M. Qin, C. Sun and D. Vucinic, "Robustness of neural networks against storage media errors," in *arXiv preprint arXiv:1709.06173*, 2017.

[13] D. S. Phatak and I. Koren, "Complete and partial fault tolerance of feedforward neural nets", *IEEE Trans. Neural Netw.*, vol. 6, no. 2, pp. 446-456, Mar. 1995.

[14] T. Haruhiko, M. Masahiko, K. Hidehiko and H. Terumine, "Enhancing both generalization and fault tolerance of multilayer neural networks," *2007 International Joint Conference on Neural Networks*, pp. 1429-1433, Aug. 2007.

[15] L. Zheng, S. Duffner, K. Idrissi, C. Garcia and A. Baskurt, "Siamese multi-layer perceptrons for dimensionality reduction and face identification," *Multimedia Tools and Applications*, vol. *75,* no. 9, pp.5055-5073, 2016.

[16] D. Yi, Z. Lei, S. Liao and S. Z. Li, "Deep Metric Learning for Person Re-identification," *2014 22nd International Conference on Pattern Recognition*, pp. 34-39, Aug. 2014.

[17] S. Liu, P. Reviriego, X. Tang, et al., "Result-based re-computation for error-tolerant classification by a support vector machine", *IEEE Transactions on Artificial Intelligence*, vol. 1, no. 1, pp. 62-73, 2020.

[18] E. M. El Mhamdi and R. Guerraoui, "When neurons fail Technical report," EPFL, Lausanne, Tech. Rep. EPFL-WORKING-217561, 2016.

[19] C.-T. Chin, K. Mehrotra, C. K. Mohan and S. Rankat, "Training techniques to obtain fault-tolerant neural networks", *Proc. 24th Int. Symp. Fault-Tolerant Comput. (FTCS) Dig. Papers*, pp. 360-369, Jun. 1994.

[20] N. Wei, S. Yang and S. Tong, "A modified learning algorithm for improving the fault tolerance of bp networks", *Proc. IEEE Int. Conf. Neural Netw.*, vol. 1, pp. 247-252, Jun. 1996.

[21] N. Kamiura, Y. Taniguchi, T. Isokawa and N. Matsui, "An improvement in weight-fault tolerance of feedforward neural networks," *Proceedings 10th Asian Test Symposium*, pp. 359-364, Aug. 2001.

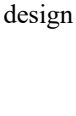
[22] T. Haruhiko, K. Hidehiko and H. Terumine, "Partially weight minimization approach for fault tolerant multilayer neural networks," *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, vol. 2, pp. 1092-1096, 2002.

[23] L. Matanaluza *et al*., "Emulating the effects of radiation-induced soft-errors for the reliability assessment of neural networks," in *IEEE Transactions on Emerging Topics in Computing*, 2021 (early access).

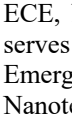
[24] IS Committee. (2008). 754–2008 IEEE standard for floating-point arithmetic. *IEEE Computer Society Std*, 2008.
[25] M. Maniatakos, M.L. Michael, Y. Makris, "Vulnerability-based interleaving for multi-bit upset (MBU) protection in modern microprocessors," in *IEEE International Test Conference*, pp.1-8, 2012.
[26] Y. LeCun, L. Bottou, Y. Bengio et al., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.
[27] H. Xiao, K. Rasul and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," in *arXiv preprint arXiv:1708.07747*, 2017.
[28] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, 2009.
[29] Y. Netzer, T. Wang, A. Coates, et al., "Reading digits in natural images with unsupervised feature learning," 2011.
[30] A. Coates, A. Ng and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011.
[31] S. Lin and D. J. Costello, Error control coding, Scarborough: Pretice hall, 2001.
[32] G. Li, S. K. S. Hari, M. Sullivan, et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications", in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1-12, 2017.
[33] R. Hadsell, S. Chopra and Y. and LeCun, "Dimensionality reduction by learning an invariant mapping," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, 2006.
[34] S.K. Venkataramanaiah, Y. Ma, S. Yin, et al, "Automatic compiler based fpga accelerator for cnn training," in *29th International Conference on Field Programmable Logic and Applications (FPL)*, pp.166-172, 2019.
[35] X. Zhang, D. Zhuang, C. Xie, et al, "Enabling highly efficient capsule networks processing through software-hardware co-design," *IEEE Transactions on Computers*, vol.70, no.4, pp.495-510, 2021.
[36] T. Luo, S. Liu, L. Li, et al, "DaDianNao: A Neural Network Supercomputer," *IEEE Transactions on Computers*, vol.66. no.1, pp.73-88, 2016.
[37] E Cai, D.C. Juan, D. Stamoulis, et al, "Neuralpower: predict and deploy energy-efficient convolutional neural networks," in *Asian Conference on Machine Learning*, pp.622-637, 2017.
[38] Y. Li and W. Chu, "Implementation of single precision floating point square root on FPGAs," in *Proceedings. The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines Cat. No. 97TB100186*, Napa Valley, CA, USA, 1997.
[39] N. Nedjah, R. M. da Silva, L. M. Mourelle, et al., "Dynamic MAC-based architecture of artificial neural networks suitable for hardware implementation on FPGAs," *Neurocomputing,* vol. 72, no. 10-12, pp. 2171-2179, 2009.
[40] S. Liu, X. Tang, F. Niknia, et al., "Stochastic dividers for low latency neural networks", *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2021 (Early access).
[41] N. B. Gaikwad, T. Varun, K. Avinash, et al., "Efficient FPGA implementation of multilayer perceptron for real-time human activity classification." *IEEE Access* 7 (2019): 26696-26706.
[42] R. Sreehari., D. Vijayasenan and M. R. Arulalan. "A Hardware Accelerator Based on Quantized Weights for Deep Neural Networks." In *Emerging Research in Electronics, Computer Science and Technology*, pp. 1079-1091. Springer, Singapore, 2019.
[43] L. D. Medus, T. Iakymchuk, J. V. Frances-Villora, et al., "A novel systolic parallel hardware architecture for the FPGA acceleration of feedforward neural networks." *IEEE Access* vol.7, pp.76084-76103, 2019.
[44] W. Isaac, X. Yang, T. Liu, et al., "FPGA acceleration on a multi-layer perceptron neural network for digit recognition." *The Journal of Supercomputing* , pp.1-18, 2021.
[45] Y. Liu, S. Liu, Y. Wang, F. Lombardi, et al., "A stochastic computational multi-layer perceptron with backward propagation." *IEEE Transactions on Computers* 67, no. 9, pp. 1273-1286, 2018.

Ziheng Wang (S'21) received the BEng degree in electronic and information engineering from Harbin Institute of Technology, Harbin, China, in 2018, and the MS degree in electrical engineering from University of Pennsylvania in 2020. He is studying for the PhD degree in the Department of Electrical and Computer Engineering, Northeastern University. His current research direction is neural networks and stochastic computing.

Farzad Niknia (S'21) received the B.Sc. and M.Sc. degrees in Electrical and Electronics Engineering from the University of Tabriz, Iran in 2014 and 2018. He worked as a research assistant at IC Design Lab through his M.Sc. degree. He is currently working towards the Ph.D. degree in Computer Engineering at Northeastern University, Boston as a research assistant with a concentration on ASIC Design. His research interests include ASIC and FPGA design, VLSI, EDA tools, design for test and hardware security.



Shanshan Liu (M'19) received the Ph.D. degree in Microelectronics and Solid-State Electronics from Harbin Institute of Technology, Harbin, China, in 2018. She was a post-doctoral researcher with the Department of Electrical and Computer Engineering (ECE), Northeastern University, Boston, USA, from 2018 to 2021, and is currently an Assistant Professor with the Klipsch School of ECE, New Mexico State University, Las Cruces, USA. She serves as an Associate Editor for the IEEE Transactions on Emerging Topics in Computing and the IEEE Transactions on Nanotechnology, a Guest Editor for the IEEE Transactions on Circuits and Systems-I, and a technical program committee member for the IEEE International Symposiums on DFT, NANO and IOLTS. Her research interests include fault tolerance design in high performance computing systems, emerging computing, VLSI design, dependable machine learning, error correction codes.



Pedro Reviriego (M'04-SM'15) received the M.Sc. and Ph.D. degrees in telecommunications engineering from the Technical University of Madrid, Madrid, Spain, in 1994 and 1997, respectively. From 1997 to 2000, he was an Engineer with Teldat, Madrid, working on router implementation. In 2000, he joined Massana to work on the development of 1000BASE-T transceivers. From 2004 to 2007, he was a Distinguished Member of Technical Staff with the LSI Corporation, working on the development of Ethernet transceivers. From 2007 to 2018 he was with Nebrija University. He is currently with Universidad Carlos III de Madrid working on high speed packet processing and fault tolerant electronics.

Paolo Montuschi (M'90-SM'07-F'14) is a professor with the Department of Computer Engineering at Politecnico di Torino, Italy, and Rector's Delegate for Information Systems. His research interests include computer arithmetic, graphics, and intelligent systems. He is a life member of the International Academy of Sciences in Turin, and of HKN, the Honor Society of IEEE. He serves as the Editor-in-Chief of the IEEE Transactions on Emerging Topics in Computing, the 2020-23, Chair of the IEEE TAB/ARC, and the co-Chair of the 2021 TAB/PSPB Ad Hoc Committee on Publications Strategy. Previously, he served in a number of positions, including the Editor-in-Chief of the IEEE Transactions on Computers (2015-18), the 2017-20 IEEE Computer Society Awards Committee Chair, a Member-at-Large of IEEE PSPB (2018-20), and as the Chair of its Strategic Planning Committee (2019-20). More information at http://staff.polito.it/paolo.montuschi.

Fabrizio Lombardi (M'81-SM'02-F'09) received the B.Sc. degree (Hons.) in electronic engineering from the University of Essex, U.K., in 1977, the master's degree in microwaves and modern optics and the Diploma degree in microwave engineering from the Microwave Research Unit, University College London, in 1978, and the Ph.D. degree from the University of London in 1982. He is currently the International Test Conference (ITC) Endowed Chair Professorship with Northeastern University, Boston, USA. His research interests are bio-inspired and nano manufacturing/computing, VLSI design, testing, and fault/defect tolerance of digital systems. He was the Editor-in-Chief of the IEEE TRANSACTIONS ON COMPUTERS from 2007 to 2010 and the inaugural Editor-in-Chief of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING from 2013 to 2017, IEEE TRANSACTIONS ON NANOTECHNOLOGY from 2014 to 2019. He is currently the Vice President for Publications of the IEEE Computer Society, the 2021 President-elect of the IEEE Nanotechnology Council.