E^{2}CNNs: Ensembles of Convolutional Neural Networks to Improve Robustness Against Memory Errors in Edge-Computing Devices

Flavio Ponzina, Miguel Peón-Quirós, Andreas Burg, *Senior Member, IEEE*, David Atienza, *Fellow, IEEE*

Abstract—To reduce energy consumption, it is possible to operate embedded systems at sub-nominal conditions (e.g., reduced voltage, limited eDRAM refresh rate) that can introduce bit errors in their memories. These errors can affect the stored values of CNN weights and activations, compromising their accuracy. In this paper, we introduce Embedded Ensemble CNNs (E^{2}CNNs), our architectural design methodology to conceive ensembles of convolutional neural networks to improve robustness against memory errors compared to a single-instance network. Ensembles of CNNs have been previously proposed to increase accuracy at the cost of replicating similar or different architectures. Unfortunately, SoA ensembles do not suit well embedded systems, in which memory and processing constraints limit the number of deployable models. Our proposed architecture solves that limitation applying SoA compression methods to produce an ensemble with the same memory requirements of the original architecture, but with improved error robustness. Then, as part of our new E^{2}CNNs design methodology, we propose a heuristic method to automate the design of the voter-based ensemble architecture that maximizes accuracy for the expected memory error rate while bounding the design effort. To evaluate the robustness of E^{2}CNNs for different error types and densities, and their ability to achieve energy savings, we propose three error models that simulate the behavior of SRAM and eDRAM operating at sub-nominal conditions. Our results show that E^{2}CNNs achieves energy savings of up to $80\,\%$ for LeNet-5, $90\,\%$ for AlexNet, $60\,\%$ for GoogLeNet, $60\,\%$ for MobileNet and $60\,\%$ for an optimized industrial CNN, while minimizing the impact on accuracy. Furthermore, the memory size can be decreased up to $54\,\%$ by reducing the number of members in the ensemble, with a more limited impact on the original accuracy than obtained through pruning alone.

Index Terms—Edge Computing, CNNs, Error Robustness, Ensemble architectures.

1 Introduction

The outstanding performance of convolutional neural networks (CNNs) in classification and detection tasks makes them the most popular machine learning architectures for a continuously growing number of applications, from biomedical health devices to autonomous driving vehicles [1], [2]. Two common methods to improve CNN accuracy are increasing the complexity of their internal structure and using ensemble architectures, in which independently trained models are combined together at inference time. The latter approach is the basis of random forests [3], but the same idea can also be used for CNNs. Thus, in a CNN ensemble, each instance is fed with the same input data and the individual predictions are combined to compute the final output. Several studies [4], [5] have shown how this approach can significantly improve accuracy by properly training a variable number of models.

The advent of edge-computing has made it common to deploy machine learning algorithms on edge devices. However, the deployment of CNNs on embedded systems with limited hardware resources and restricted energy budgets struggles with two main obstacles. On one side, their limited storage capacity hampers the use of large models; on the other side, their reduced computing power can be a limiting factor for real-time applications.

The energy efficiency of embedded systems can be improved with techniques such as dynamic voltage and frequency scaling (DVFS). However, although logic circuits can function properly (albeit with longer delays) at reduced voltage levels, the data integrity of SRAMs is particularly sensitive to voltage scaling, which may lead to the emergence of memory errors. A similar problem arises with eDRAMs, for which the refresh process represents on average $5\,\%$ of the total number of memory operations (see Section 4.1). To decrease their energy consumption, the refresh rate can be reduced, but this increases the risk of bit flips. In addition to these considerations, edge devices may also be exposed to harsh environments that introduce errors in the memory contents even at nominal conditions.

In traditional applications that require perfect memory retention, the aforementioned factors place stringent limits on the energy savings that can be obtained. Memory in-

- *This work was supported in part by the Swiss NSF ML-Edge Project under Grant 200020_182009, and by the ERC Consolidator Grant COMPUSAPIEN (GA No. 725657), and the EC H2020 WiPLASH (GA No. 863337) project.*

- *F. Ponzina, M. Peón-Quirós and D. Atienza are with the Embedded Systems Laboratory (ESL), EPFL, Lausanne, Switzerland. E-mail: flavio.ponzina@epfl.ch, miguel.peon@epfl.ch, and david.atienza@epfl.ch*

- *A. Burg is with the Telecommunications Circuits Laboratory (TCL), EPFL, Lausanne, Switzerland. E-mail: andreas.burg@epfl.ch*

tegrity techniques such as error-correcting code (ECC) are adequate to protect against natural error causes, but they introduce considerable overheads in terms of area and energy consumption that reduce the energy efficiency of the system. Additionally, in conditions in which several errors may affect a single memory word, more complex mechanisms with further energy overheads may be required.

In this work, we build upon the natural robustness of CNNs against memory errors that has been observed in previous works to propose Embedded Ensemble CNNs (E^{2}CNNs), a new design methodology that opens the way to additional energy savings in embedded devices implementing CNNs without sacrificing their accuracy. The main contributions of this paper are the following:

- We introduce E^{2}CNNs, a design methodology that increases error robustness of CNNs by using ensemble architectures with no additional memory requirements compared to an original single instance CNN.
- As part of the E^{2}CNNs methodology, we propose a heuristic method to drive the design of the ensemble given an expected error density in the memory.
- We define three error models to simulate the behavior of SRAM and eDRAM operating at sub-nominal conditions. We then show how the proposed E^{2}CNNs architecture is more robust against these types of memory errors than the original models; hence, we demonstrate that E^{2}CNNs can be used to save energy in edge devices running CNNs while reducing the impact on accuracy.
- We conduct an analysis of the error resiliency of weights and activations in fixed-point quantized CNNs. The results of this study can be applied to E^{2}CNNs to further reduce energy consumption with minimum impact on accuracy.
- We show how E^{2}CNNs improves accuracy in comparison to pruning alone for the same accuracy level.
- We apply E^{2}CNNs on three well-known CNN models (LeNet-5, AlexNet and GoogLeNet) and a tightly-optimized industrial one. We use these results to build our heuristic, which is then evaluated on an additional model (MobileNet).

The rest of this paper is organized as follows. First, we review the related works in Section 2. Then, in Section 3 we present E^{2}CNNs, whose introduction represents the main contribution of our work. We also introduce a heuristic procedure to automate the design of E^{2}CNNs for expected memory error rates. In Section 4 we describe our experimental methodology and we discuss our results in Section 5. Finally, we draw our conclusions in Section 6.

2 Background and Motivation

2.1 Convolutional neural networks

In their simplest form, CNNs consist of linear and non-linear layers, sequentially connected. Convolutional layers allow the model to change the input data dimensionality, extracting specific features from the incoming data. Moreover, they are also particularly useful for image data inputs, as the convolution is a translation-invariant operation. On the other hand, non linear layers, such as the rectifier unit (ReLU), allow the model to extract more complex structures from the input. The interest in having always more accurate architectures led to the development of complex models. In particular, deep neural networks (DNNs) are characterized by an increased number of hidden layers. Other recent improvements come from the inception layers and residual blocks, particularly useful during the training phase to reach higher precision and better generalization capabilities.

2.2 Ensembles of convolutional neural networks

Though ensemble methods were originally proposed many years ago, they still attract the interest of the research community because of their potential. During the learning phase of an ensemble, several models are independently trained on their target dataset. Then, their individual predictions are collected and processed to estimate the best output during online inference. The way in which individual predictions are handled in ensembles is a deeply discussed topic in machine learning. The simplest approach is to average the predictions of individual voters, while other studies combine them in more complex layers [6], [7]. These studies also show how ensemble architectures provide better generalization capabilities compared to single-instance approaches. The number of architectures deployed to form the ensemble and the way to manage their output is not fixed. In the literature, we find ensembles made up of few architectures, while other studies propose the adoption of tens or hundreds of different models. The number of models composing the ensemble has a direct impact on its applicability in real life applications. In particular, the larger the ensemble, the higher the offline training time and the memory requirements at run-time. Moreover, the online performance is also affected, as more inference steps have to be executed. One interesting approach to reduce the overhead of large ensembles is presented in [8], where a data augmentation stage returns N different images, which are then classified by N different CNN instances. Although the authors propose a staged process that checks the obtained confidence before consecutively activating additional instances, their solution still increases the energy and resource requirements of the original network. In contrast, E^{2}CNNs aims at improving resilience against errors without creating more pressure on the limited resources of embedded devices.

2.3 Pruning

Pruning is a well known approach that reduces the size of a CNN [9]. In particular, this approach aims at removing some of the model parameters and it typically comes in two flavors: fine-grain and coarse-grain. The former method selectively removes individual weights from the model, while coarse-grain pruning removes entire filters, significantly reducing the model size at each step. In both cases, the goal is to obtain a compressed model in which the drop in accuracy given by the reduction in the trainable parameters is as contained as possible. Thus, different methodologies and selection criteria have been proposed [10], [11]. Usually, the pruning procedure is jointly run with a fine-tuning step, in order to recover part of the accuracy drop. We use coarse-grain pruning to build the instances of our ensembles.

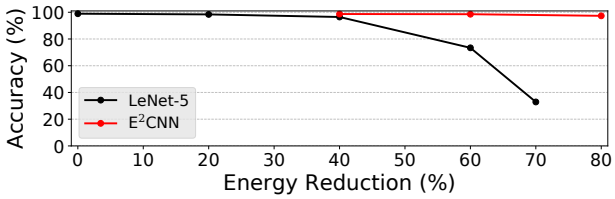


Fig. 1. Accuracy at different sub-nominal operating conditions for increasing energy savings. Our proposal better preserves the initial accuracy by improving the robustness against memory errors (SRAM).

2.4 Quantization

Fixed-point quantization uses a reduced bitwidth to represent the model parameters and activations [12]. Under this perspective, different quantized representations are possible: the most popular are FXP16 and FXP8, in which the model parameters are represented on 16 and 8 bits, respectively. A fixed-point representation on $B > 1$ bits is characterized by the position of the decimal point. It determines the number of bits used for the integer and decimal parts, and directly influences the data range and the smallest representable values.

In our work, we apply model-dataset specific quantization as it improves robustness against errors with respect to floating-point. We also adjust the number of integer bits to limit the impact of errors on the absolute value of the numbers represented. The benefits of these techniques have already been reported in the literature [13], [14] and our own experiments show that they provide equivalent improvements to the clipping of "implausible values" proposed in [15]. Furthermore, this decision reduces significantly the memory and energy requirements of the original CNN model with limited impact on accuracy.

2.5 Unreliable operations in memories

Errors in memories can catastrophically affect the function of a system. In general, errors may appear as a result of several reasons. On one side, defective memory locations may occur during the manufacturing process. On the other side, errors in memory may also appear as a consequence of particular environmental conditions: radiation particles may interact with the memory bit cells, producing flips in the stored logic values (i.e., single event upsets). These conditions may be common in avionics, space or physics applications. However, memory errors can also be the consequence of energy reduction techniques, which may represent a common scenario for edge nodes. Working at sub-nominal conditions can reduce energy consumption; different approaches have been proposed with particular focus on the memory subsystem. As example, operating SRAMs at a lower voltage reduces the energy cost due to cheaper memory accesses. Unfortunately, the lower voltage may introduce permanent errors in the weakest memory cells, which become unable to flip their content. Similarly, when the refresh rate of eDRAMs is reduced to save energy, the capacitor discharges and the cell loses its contents.

The resilience of CNNs to memory errors has been explored in [14], with the conclusion that they can tolerate relatively high bit error rates with a small drop in accuracy. This observation enables more aggressive power saving techniques that would be unacceptable in other domains.

Previous works propose injecting Gaussian noise in the input image [16] or in the intermediate layers [17] to simulate environmental conditions during the training procedure and obtain a more robust architecture. However, Gaussian noise injection does not suite well our context: a normal distributed noise disproportionately affects the memory bits (i.e., least-significant bits (LSBs) have a higher probability of being affected compared to most-significant bits (MSBs)), while faulty bit cells may randomly appear inside the memory chip. Curricular retraining seems to suite better our case, as it injects simulated memory errors at a progressive rate during training. This method is proposed in [15] alongside mapping of CNN data types (i.e., coefficients and buffers of each layer) on memory partitions to create additional robustness at inference time. However, to be effective, this approach must be aware of the memory error distribution in the specific target device to match the error tolerance of the CNN components with the error rate of memory partitions. In comparison, our solution is more general, as it can be efficiently applied to different error models without requiring knowledge on the underlying hardware. Our experiments (c.f., Section 5.4) show that E^{2}CNNs increases in some cases the error robustness of the network by a factor of $10\times$, which is comparable to the results presented in [15] for curricular retraining. Although outside the scope of this work, future works could explore how to combine both techniques to compound their benefits.

As motivational example for this work, Fig. 1 shows the accuracy of LeNet-5 for different energy savings, obtained by operating the memory at sub-nominal conditions, with a consequent memory error density. We observe that the original implementation has some degree of intrinsic error tolerance that allows saving up to $40\,\%$ of energy in the memory subsystem without significant impact on accuracy. However, the solution obtained through E^{2}CNNs can save up to $80\,\%$ energy, for an equivalent level of accuracy.

3 E^{2}CNNs: Embedded Ensemble CNNs Design Methodology

CNNs have demonstrated to be quite resilient to noise [18]. However, their resiliency is affected by factors like memory error rate, data representation, model size and structure. In particular, the same number of errors in memory can differently affect accuracy, according to the adopted data representation [14]. As example, floating-point formats make CNNs particularly sensitive to memory errors, because errors in the exponent bits may induce huge changes in the values or create Not-a-Number (NaN) values. On the contrary, fixed-point representations are more robust, in particular when a reduced number of integer bits is used, limiting the representable dynamic range. Finally, even the CNN structure affects error tolerance. In fact, networks oversized for their target dataset are more robust, because their redundancy limits the effects of individual errors.

We propose a solution based on ensembles of CNNs to increase the robustness of embedded systems in these conditions. The increased robustness can positively impact

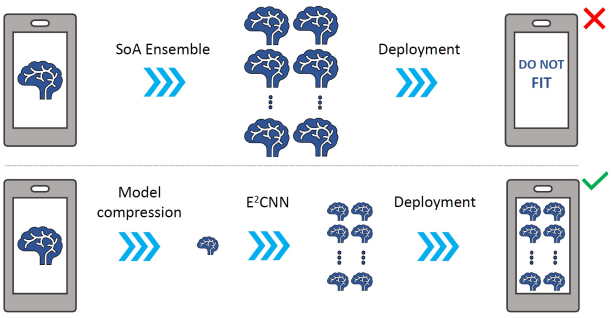


Fig. 2. Traditional ensemble (top) compared to the E²CNNs architecture (bottom). In E²CNNs, the initial model is first compressed and then replicated several times to build up an ensemble that meets the same memory requirements of the original model.

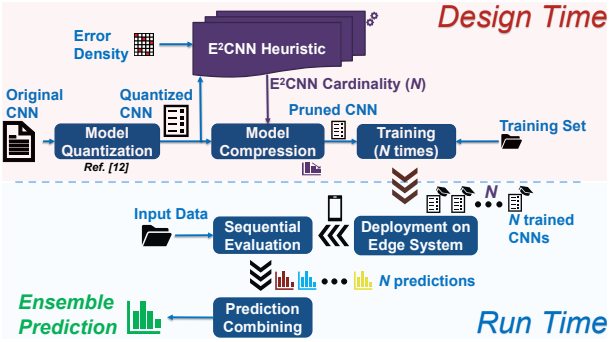


Fig. 3. E²CNNs design and use. At design time, the ensemble is built by training several times a pruned CNN. Then, the individual predictions are averaged together to compute the ensemble output at inference time.

the energy efficiency of the target system, because more vigorous low power techniques can be applied. Alternatively, similar benefits can be obtained for systems deployed in noisy environments.

3.1 E²CNNs architecture

Embedded Ensemble CNNs (E²CNNs) includes a new ensemble-based CNN architecture, which is our proposal to improve error tolerance in embedded systems running convolutional networks. Although state-of-the-art (SoA) ensembles may be difficult to deploy on edge devices due to their high memory and energy requirements, E²CNNs are specifically designed for this purpose. Our proposed E²CNNs architecture applies SoA compression techniques to reduce the size of the initial CNN in two steps: in first place, the target convolutional model is quantized by using a fixed-point representation. As this step directly impacts the achievable accuracy, we try to compress the model by reducing the bitwidth as much as possible, while limiting the impact on the original accuracy. If the original model is already quantized, then this is the starting point. In second place, we apply pruning to further compress the model to $1/N$ of its initial size. This step can be run in different ways, depending on the pruning algorithm and filter removal criteria. The result is a compressed model that has to be retrained to recover from the possible drop in accuracy. Therefore, if N is the compression ratio, then we train independently the obtained CNN N times. Finally, we deploy the N trained models on the target device and run them as an ensemble. The output class probabilities are then averaged together to compute the ensemble prediction.

The main difference with traditional proposed ensemble solutions is shown in Fig. 2: while simply replicating the same model could prevent the deployment on a memory constrained device, with E²CNNs there is no overhead in terms of memory occupation, which is approximately the same as in the original case. Although our solution suggests to deploy N instances to build the ensemble for a hypothetical N-E²CNNs, this is obviously not mandatory. Ensembles with $M > N$ models will increase both robustness and memory requirements. On the other hand, memory footprint, energy cost and computational performance can be also improved by building an ensemble with $M < N$ voters. Thus, we also explore this possibility in Section 5.

At inference time, the output of E²CNNs is computed by averaging the individual predictions of its instances. Although traditional ensembles may also use a majority voting approach to predict the output class, computing the average of the individual instances' output class probabilities has two main advantages. On one side, it enables the use of ensembles with just two instances, while this configuration could not be used in a voting-based ensemble. On the other side, averaging together the outputs of the deployed CNNs allows the ensemble to achieve higher accuracies, as we will discuss in Section 5.3. The methodology used to build and use E²CNNs is depicted in Fig. 3.

3.2 Model compression

Model compression is used to obtain a smaller CNN from the original architecture. As discussed in Section 3.1, our solution requires a compression ratio of N, so that the resulting ensemble of N instances does not produce memory overheads with respect to the single instance model. We apply coarse-grain pruning, in which entire filters are removed from the model, to compress the original memory occupation of the convolutional model to $1/N$ of its initial size. Removing a whole filter reduces the size of the output feature and consequently, the following kernel matrix is reduced as well. The pruning step is run within an iterative procedure: one filter is removed at a time, until the desired compression rate is reached. At each iteration, a filter is randomly selected across the convolutional layers and removed from the CNN. This random choice allows the original architecture to be pruned without the need for a pre-trained model and has been shown to provide similar results when compared to [19], where filters are removed based on their absolute average magnitude.

The adopted procedure takes only the model memory occupation as a constraint. The main result is that, while the memory footprint is reduced by a factor of N, the corresponding number of MAC operations may not be equally reduced. Thus, potential overheads can occur during the ensemble execution. In our experiments, this effect becomes relevant only in LeNet-5, because its small size produces an unbalanced pruning. On the contrary, this behavior does not produce significant effects in larger models. Future work can also consider the number of MACs.

3.3 Selecting the number of instances in E²CNNs

To obtain an ensemble with similar memory footprint as the original CNN, we define N as both the model compression ratio and the number of deployed instances. Therefore, we propose an approach to adjust N for the expected memory error rates.

A full exploration would need to examine many different compression ratios and, for each of them, the original model would need to be compressed and then trained N times. This option can easily become computationally prohibitive. On the opposite, an analytical solution may provide optimal results, but such an approach would need to consider the CNN structure, the complexity of the input data, the pruning algorithm, the retraining strategy and the expected error rate. In that direction, [13] analyzes the factors that contribute to CNN resiliency against soft errors, highlighting the many variables involved. Furthermore, an analytical approach in our context would need to consider as well the error type and the memory working parameters. As the resulting design space would be too large to be exhaustively explored in a reasonable time, we propose a heuristic to search through it and guide the selection of a suitable ensemble architecture (i.e., the number of instances) given an expected memory error rate. In this way, our methodology reduces the effort to build a proper ensemble, by minimizing the number of model trainings and evaluations required.

We propose a two step approach (Algorithm 1). First, we pick a quantized representation that does not lead to a significant drop in accuracy. Based on previous observations, we suggest a quantized data representation that minimizes the size of the integer part, which positively impacts the final error tolerance. Then, the resulting quantized model is iteratively pruned, trained and tested. Given the considerable effort required to train a model, we propose a logarithmic search to speed up this step. At each iteration we prune the previously explored architecture by a factor of two. The loop is controlled by a threshold for the accuracy drop. Our preliminary experiments suggest setting the maximum drop to $5\,\%$ in case of high error rates, while a more conservative drop of $1\,\%$ may allow the designer to achieve a higher final accuracy in the case of low expected error rates. The benefits of the proposed solution are twofold. First, the logarithmic search dramatically reduces the number of solutions to explore. Second, it allows to select a proper architecture based on the expected environmental conditions.

The aforementioned heuristic method is the result of a full exploration analysis performed on four benchmarks. These results are presented in Section 5. Then, we evaluate its prediction on an additional benchmark. Although there may be cases in which the heuristic outcome is not strictly optimal, its searching criteria, based on the accuracy of a single voter of the ensemble architecture, dramatically reduces the computational cost of the entire procedure, since it only requires to train one CNN, regardless of the cardinality of the ensemble under analysis. In particular, Algorithm 1 is executed only once for a given benchmark: it suggests the proper E²CNNs cardinality for a certain error rate based on the analysis of an individual (pruned) CNN. Then, all the instances in the ensemble share the same architecture and only need to be independently trained.

Algorithm 1 E²CNNs heuristic

```
1:  procedure BUILDER(ErrorRate, QuantLevels, AccMin)
2:      i ← 0
3:      repeat
4:          Quant ← QuantLevels[i]
5:          NewModel ← Quantize(Model, Quant)
6:          if NewModel.Acc ≥ AccMin then
7:              Model ← NewModel
8:          i++
9:      until NewModel.Acc < AccMin
10:     Accuracy ← Model.Acc
11:     if ErrorRate >= 0.0001 then
12:         DropThreshold ← 5%
13:     else
14:         DropThreshold ← 1%
15:     repeat
16:         NewModel = Model.prune(compression = 2x)
17:         NewModel.Train()
18:         drop ← Accuracy − NewModel.Acc
19:         if drop < DropThreshold then
20:             Model ← NewModel
21:         i++
22:     until Drop ≥ DropThreshold
```

4 Experimental Setup

In this section, we describe how we conducted our experiments. We start with a brief description of the target embedded architecture. Then, we present the inference simulation process and the way error injection has been designed. We present the selected benchmarks, with a description of how they have been trained and quantized. Finally, we explain our experimental methodology.

4.1 Target device architecture

To estimate the performance and energy improvements of our solutions with respect to the baseline CNN, we define a hardware architecture model in which our benchmarks are executed. Thus, we consider a system composed of a processor directly connected to the main memory by means of a 32 bit bus, without any cache hierarchy. We divide our memory into three logical blocks. The first one is used to store the input image, which may be updated in memory from a peripheral camera through a DMA transfer while the previous image is being analyzed. The second one stores the model parameters. Finally, a third block is used to store two buffers for the intermediate activation maps.

At the application level, we do not assume the use of SIMD processing. For every MAC operation, the required activation and weight values are read from memory. However, activations use 16 bits, while the weights are stored in 8 bits. This means that two activations and four model parameters can be transferred over the 32 bit bus in each memory operation. We assume this behavior in our experiments. Additionally, partial results of convolutions are stored in local registers and only the final output is written back to memory. For eDRAM, we consider a typical organization in banks of 256 words of 32 bits working at $500\,\mathrm{MHz}$ [20]. We assume that the processor accesses the memory every cycle, that read and write operations are executed in one cycle, and that all the memory banks are refreshed in parallel. Under these conditions, we estimate that the refresh operations account on average for a $5\,\%$ overhead on the execution time. However, their impact on energy consumption can be

considerably higher, because a refresh cycle always accesses multiple banks in parallel (as opposed to a regular access cycle, which targets a single bank). Hence, refresh can easily dominate the memory power consumption, even with a negligible impact on access bandwidth.

When evaluating the execution of E^{2}CNNs, the different members of the ensemble are executed one after the other, thus reducing the total memory footprint with respect to the original model. As each member is trained independently, the same faulty memory positions affect each of them in different ways.

4.2 Inference simulation

We simulate the execution of embedded devices on multi-core servers to efficiently analyze whole chip populations on different models, input datasets and bit error probabilities. Our simulations focus on the virtualization of the main memory, as we are interested in evaluating the system resiliency to memory errors. Hence, we have built a framework to simulate the injection of memory errors based on template classes that wrap basic types and control their positioning on a memory address space. Then, we implement our solver in C++ using these wrappers. The solver has been developed to mimic real embedded systems implementations. For example, it minimizes data movements between buffers and reuses continuously the same two buffers across layers. In this way, it ensures that the footprint of the application is bounded by the footprint required at the largest CNN layer. The same layers' implementation is used in different architectures. We fuse the non linear function (e.g., ReLU) with the execution of the previous convolutional layer, thus minimizing data movements because the accumulated outputs are stored in local registers. This limits the impact of memory errors on the intermediate values, as it would happen in practice in real implementations. Pooling layers are independently executed, reading the input values from the main memory.

4.3 Memory emulation and error injection

We develop three different error injection processes to simulate errors appearing in SRAM and eDRAM as a consequence of sub-nominal operating conditions. In our experiments, the memory is composed of a safe array, whose data is protected, as well as an unsafe array, in which errors may appear. The size of the two arrays can be arbitrarily set, allowing us to simulate errors affecting different areas of the memory. This method allows us to extract additional information such as the number of accesses, read or write operations to each memory word.

4.3.1 Injection of errors in SRAM

We define errors affecting SRAM as faulty bit cells permanently stuck at a given logic value, either 0 or 1. For banks operating at sub-nominal voltage, errors in the peripheral circuitry surrounding the memory array tend to produce correlated errors. In particular, the sense amplifiers (SAs) are sensitive to voltage variations and can produce errors in entire bitlines (BLs). However, the SAs and other elements can be designed to increase their robustness, leaving only the errors due to the variability in the bit-cells themselves,

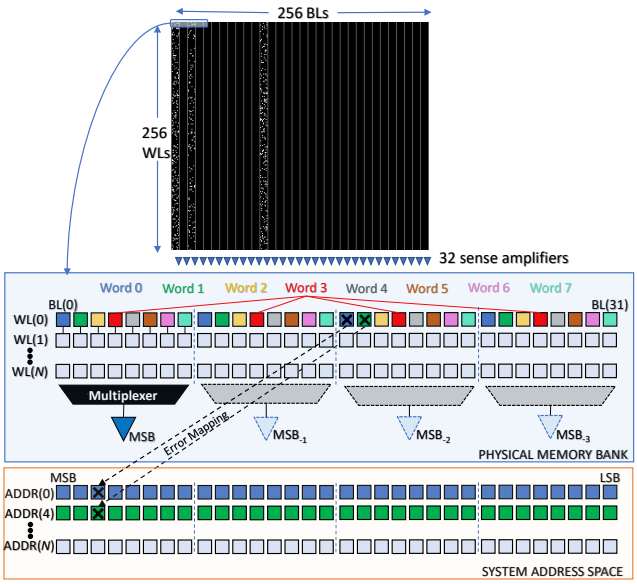


Fig. 4. Top: SRAM bank layout for injection of correlated errors. Each sense amplifier is connected to 8 BLs; each WL contains eight 32-bit words. Bottom: Bit interleaving configuration and corresponding mapping into the system address space. A single disturbed sense amplifier increases the probability of error in the cells connected to it, which translates into errors in the same bit (weight) of the words contained in the bank (eight words per WL, 256 WLs). Multiple failing sense amplifiers in one bank generate multi-bit errors in the contained words.

which then follow a uniform distribution in the memory matrix. In both cases, write operations fail if the supplied voltage is insufficient to provide weak bit cells with enough current to flip their content. These faults can be considered permanent errors that appear in SRAMs when operating at a low voltage. The number of errors is a direct consequence of the applied voltage in an SRAM device. However, their distribution in the memory differs from one chip to another as a result of manufacturing process variability.

To reflect the aforementioned situations, in this work we consider both correlated and uniformly distributed errors. In the former case, correlation along BLs increases the probability that different memory words fail in the same bit position. As this effect is linked to the physical memory layout, we consider the SRAM configuration shown in Fig. 4, which consists of a folded matrix composed of 256 BLs and 256 wordlines (WLs) to produce a bank of 2048 words of 32 bits. We consider a low-order interleaved memory organization in which bits of consecutive BLs correspond to the same bit position of different memory words. Using this internal bank organization, we then translate bit errors in each bank into errors in the system address space.

For static errors, the number of faults to inject in memory is computed offline (i.e., before running the inference), according to the desired error rate. Subsequently, errors are injected as "stuck-at-1" or "stuck-at-0" faults directly in the selected bits of the address space. Finally, the rate of soft errors also increases at low operating voltages, producing temporary bit flips in the memory array. In this work, we do not specifically introduce soft errors in our experiments, but the high error rates considered represent a worst case scenario, in which the number of static errors is orders of magnitude larger than the number of soft errors.

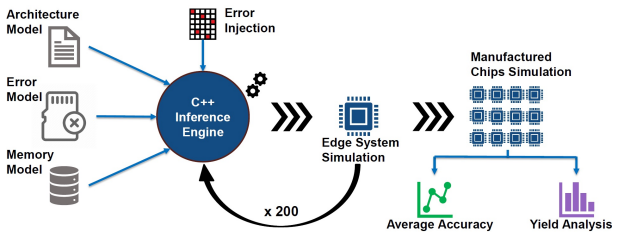


Fig. 5. Experimental methodology. The evaluation of specific architecture, error and memory models with different random error masks simulates the characteristics of an entire population of chips.

4.3.2 *Injection of errors in eDRAM*

Errors affecting eDRAMs cause temporary faults in the memory subsystem. In particular, these errors do not constantly affect specific memory locations, but occur at specific moments in time. This is what happens when the refresh rate is relaxed to reduce the number of memory accesses, hence reducing the energy cost. When the refresh rate is relaxed to reduce the number of memory accesses, due to leakage currents, the capacitors inside the bit cells discharge and the stored values may be lost without timely refresh operations. The data retention time (DRT) varies across the memory cells as a consequence of the manufacturing process variability and, in general, the worst case DRT is considered to determine the required refresh rate that prevents memory data loss.

As described in [21], the DRT of the bit cells in a generic eDRAM chip follows a log-normal distribution. This means that the number of bit cells failing when slightly reducing the refresh period is limited. However, more vigorous refresh rate reductions increase the number of failing bit cells. More precisely, leakage currents affect both logic levels. However, by design, one level is orders of magnitude more sensitive than the other. Therefore, we assume only failures which set to 0 the logic value of the affected bit cells.

In our eDRAM emulator, we assign every memory bit cell a data retention time, randomly computed according to a log-normal distribution. Then, we simulate the passage of time by counting the number of memory accesses and we introduce errors when a time longer than the bit cell data retention is passed.

4.4 Experimental methodology

We evaluate the original CNNs and the corresponding E^{2}CNNs-based alternatives for different memory error densities. However, the same error density may affect the system in different measures, depending on the errors' exact location. For example, when operating SRAMs at reduced voltages, weakest cells fail sooner than others; the bit cell error probability at each sub-nominal voltage is subjected to CMOS process variability conditions, resulting in different error maps in the manufactured device. To accurately represent the natural variability of chip populations, we evaluate the same combination of CNN model, ensemble configuration and error rate 200 times, using each time a randomly computed error map. This methodology (depicted in Fig. 5) allows us to conduct a yield analysis similar to the one reported in [20]. For the purpose of verifying our proposal, we first run experiments for the single-instance model and then we systematically evaluate ensembles of N models ($N = 1, 2, 4, \ldots$), until the resulting accuracy becomes unacceptable (i.e., lower than $10\,\%$). However, the system designer that uses our methodology can use the heuristic explained in Section 3 to minimize the number of training and testing steps required. In this line, we apply our heuristic to MobileNet and compare the performance of the selected ensemble with the other possible alternatives.

4.5 Benchmarks

We evaluate E^{2}CNNs on several models and datasets:

- LeNet-5 [22] on the MNIST dataset [23].
- AlexNet [24] on the CIFAR-10 dataset [25].
- An optimized industrial embedded CNN application for recognition of coffee capsules ("Industrial") [14].
- GoogLeNet [26] on the EuroSAT dataset [27].
- MobileNet [28] on the EuroSAT dataset [27].

TABLE 1
Relative memory footprint of input, weight and activation values, for each quantized CNN model

Model	Input (%)	Weights (%)	Activations (%)
LeNet-5	2.68	80.81	16.51
Industrial	1.92	24.66	73.42
AlexNet	0.04	94.90	5.06
GoogLeNet	0.47	90.48	9.05
MobileNet	5.94	48.51	45.55

The selected models present a wide range of structures: LeNet-5 is a small model, but still its few convolutional layers obtain excellent accuracy on the MNIST dataset. Industrial is optimized for the recognition of a specific type of objects; we picked this example because the size of the model is already tightly adjusted to the size of its dataset, which allows us to explore the limits of E^{2}CNNs. Additionally, we considered GoogLeNet to present our methodology on a relatively recent architecture whose internal structure is quite different from the previous ones. MobileNet is a CNN specifically designed for embedded systems and in this work it is used to evaluate the prediction of our heuristic.

We use our memory emulator to build a CNN solver in which we implement all the models. The accuracy of our trained error-free models on their target datasets is aligned with other published SoA results when using a floating-point representation. The proposed benchmarks have also different memory requirements, which allows us to conduct experiments both on small and large models. However, the most interesting aspect is the difference in the percentage of memory devoted to store the weights and the activations, which vary significantly across CNN models, as presented in Table 1. As an example, almost $95\,\%$ of the memory is used to store weights in AlexNet, while in Industrial this percentage is below $25\,\%$.

4.6 Training using Fake Quantization

We train our models using a floating-point representation for a variable number of epochs. Then, during the last

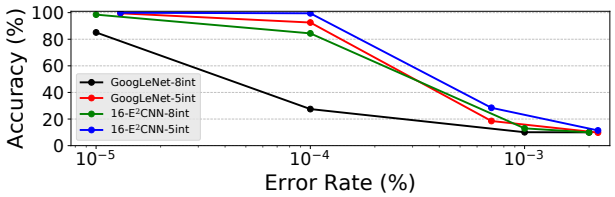


Fig. 6. Accuracy at different error rates with activations on 16 bits, but different integer bitwidths. Using unnecessarily additional 3 integer bits severely affects the CNN robustness against errors.

10 training epochs, we apply quantization to the model weights, biases and features. This approach significantly speeds up the training process since just 10 epochs of quantized training are enough to recover from the accuracy drop caused by the quantization. The entire procedure is executed in TensorFlow, using GPU-compatible code. We use the TensorFlow functionality for "Fake Quantization" [29], in which each quantized weight or activation assumes the value it would take if the desired fixed-point representation was applied. However, this representation still uses floating-point values and the TensorFlow built-in arithmetic functions are consequently still executed in floating-point. Nonetheless, in order to accurately represent the conditions in a real embedded device limited to fixed-point arithmetic, our CNN solver executes all its arithmetic operations in fixed-point. Hence, we find a negligible difference (i.e., lower than $0.1\,\%$) between the accuracy reported by TensorFlow and our implementation—which corresponds to the expected behavior of the target embedded platform.

All the models are quantized using a uniform quantization on 8 bits for the weights and a uniform quantization on 16 bits for the activations. In particular, for the model parameters we use 1 sign bit and $B-1$ bits for the decimal part (with no integer bits). Regarding activations, each model is trained using the smallest number of integer bits for which we do not observe a drop in accuracy. This decision is based on previous research that has shown that increasing the size of the representation and, in particular, the number of integer bits, reduces the robustness of the CNN against errors, because for wider representations, errors in MSBs introduce larger deviations in the arithmetical computations [14], [30]. As an alternative, mixed-precision quantization can further reduce the model size, and could also represent a viable solution to increase error resiliency with limited impact in accuracy. In this work, however, we only consider uniform quantization, but consequently mixed-precision techniques can be adopted as a complementary approach to E^{2}CNNs.

Fig. 6 shows the accuracy of GoogLeNet and a 16-E^{2}CNNs solution for different error rates. All the models use 16 bits to store the activations. However, in one case, 8 bits are used to store the integer bits (black and green curves), while in the other case, they are stored using only 5 bits (red and blue curves). The selection of the correct fixed-point representation has a crucial impact on the robustness of the CNN against memory errors. The figure also shows that, for a given fixed-point representation, E^{2}CNNs can still improve the robustness. According to this observation, in Section 5, we conduct our experiments using the best data representation that we identify for each benchmark.

To determine the optimal fixed-point representation for each model and dataset, we run an analysis on the activations distribution to understand how many bits are needed for the decimal part for a random sample of 50 input images. Based on this analysis, we determine, for each benchmark, the smallest number of integer bits able to represent the entire data range. In particular, we split the 16 bits of the activation values as follows: 1 sign bit and 4 integer bits for LeNet-5, AlexNet and MobileNet, while we use 5 integer bits for GoogLeNet and Industrial. Then, the remaining bits (i.e., 11 and 10 bits, respectively) represent the fractional part. Due to this adaptation in the number of bits, we only observe a degradation of accuracy lower than $0.1\,\%$ comparing the floating-point model with the quantized one.

5 Experimental Results

Our analysis is organized as follows: we first present preliminary findings from our initial experiments in Section 5.1 and Section 5.2. We evaluate the E^{2}CNNs architecture in the absence of memory errors in Section 5.3. Then, we conduct an analysis on edge devices equipped with SRAM in Section 5.4. In particular, we demonstrate that our proposed solution is able to improve robustness against memory errors resulting from sub-nominal operating conditions. This allows our solution to achieve larger energy savings than the original CNN, without sacrificing accuracy. Next, in Section 5.5, we propose a similar analysis for systems equipped with eDRAM. In both cases, we also show how E^{2}CNNs can be used to support additional energy saving techniques. Finally, we assess the effectiveness of the proposed heuristic method on MobileNet, in Section 5.6.

5.1 Error sensitivity of weights and activations

To investigate the resiliency of different data structures, we conduct three tests in which we inject errors in the entire data values, only in the weights/bias and only in the activations. The last two cases represent the situation in which a fraction of the memory is protected. Our results indicate that protecting the activations is crucial to preserve the accuracy in noisy environments. This observation is in contrast to [30], where the activations are found to be more robust. However, some differences between the two works may explain this opposite finding. On one hand, we quantize the model parameters on 8 bits, while the authors of ARES use 16 bits. The increased bitwidth makes errors affecting the MSBs more critical as they produce larger variations. Additionally, they run inference quantizing weights and activations only between layers, with internal operations still executed in full precision. On the contrary, we use fixed-point arithmetic during the entire process.

Our analysis reveals that the higher sensitivity of activations seems to be a common property among all tested benchmarks, without correlation with their structure. In fact, comparing our results with the architectural structures shown in Table 1, we observe that the same behavior is common both in models with very large activations (e.g., in Industrial they occupy more than $70\,\%$ of the memory) and in models in which the parameters occupy the largest percentage in memory (e.g., AlexNet requires less than $10\,\%$

TABLE 2
Contribution of input, weight and activation values to the memory footprint in the original model and E^{2}CNNs ($M = N$). In parenthesis, the percentage change in safe memory footprint. The number of MAC operations reflects the impact of E^{2}CNNs on computational cost

Model	Input (%)	Weights (%)	Activations (%)	MACs (%)
LeNet-5	2.68	80.81	16.51 (+0.00 %)	100.0
LeNet-5 2-E^{2}CNNs	2.69	80.71	16.60 (+0.54 %)	185.1
LeNet-5 4-E^{2}CNNs	2.72	80.52	16.76 (+1.51 %)	255.4
LeNet-5 8-E^{2}CNNs	3.04	90.54	6.42 (-61.11 %)	146.7
Industrial	1.92	24.66	73.42	100.0
Industrial 2-E^{2}CNNs	2.03	24.70	73.27 (-0.21 %)	112.9
Industrial 4-E^{2}CNNs	2.16	27.47	70.37 (-4.15 %)	108.1
AlexNet	0.04	94.90	5.06	100.0
AlexNet 2-E^{2}CNNs	0.04	95.88	4.08 (-19.37 %)	106.6
AlexNet 4-E^{2}CNNs	0.04	97.08	2.88 (-43.08 %)	100.3
AlexNet 8-E^{2}CNNs	0.04	99.12	0.84 (-83.40 %)	19.8
GoogLeNet	0.47	90.48	9.05	100.0
GoogLeNet 2-E^{2}CNNs	0.49	92.89	6.62 (-26.85 %)	100.8
GoogLeNet 4-E^{2}CNNs	0.50	94.94	4.56 (-49.61 %)	100.7
GoogLeNet 8-E^{2}CNNs	0.50	96.18	3.32 (-63.31 %)	100.1
GoogLeNet 16-E^{2}CNNs	0.51	97.12	2.37 (-73.81 %)	109.1
MobileNet	5.94	48.51	45.55	100.0
MobileNet 2-E^{2}CNNs	6.37	51.99	41.64 (-3.91 %)	102.6
MobileNet 4-E^{2}CNNs	6.82	55.68	37.50 (-8.05 %)	89.9
MobileNet 8-E^{2}CNNs	7.69	62.82	29.49 (-16.06 %)	79.2

of the memory for the activation buffers). To protect the required memory banks against errors, different actions are needed, such as, either using the nominal voltage in SRAMs or the reference refresh rate in eDRAMs or deploying shielded devices in case of radiation noise [31], [20], [32]. In any case, these actions result in an energy penalty or additional cost. As explained in the next section, E^{2}CNNs helps also to reduce the amount of memory to be protected.

5.2 Reduction of activation buffers using E^{2}CNNs

The adoption of E^{2}CNNs provides two additional benefits. On one side, it further reduces the memory requirements, thus enabling the desired architecture to fit tightly constrained devices. In particular, this can be done by building the ensemble with $M < N$ instances, where N represents the compression applied to the original CNN. The amount of memory savings is linearly proportional to the number of deployed instances, and it is also affected by the internal structure of the original CNN. In fact, models like Industrial have most of the memory occupied by the activation buffers (Table 1). Therefore, reducing the number of deployed instances in the ensemble marginally affects the total memory occupation, as this technique predominantly reduces the size of the memory storing the model parameters. This solution suits well models like LeNet-5, AlexNet and GoogLeNet, for which the memory footprint reduction is almost proportional to M/N. Moreover, E^{2}CNNs with $M < N$ instances not only reduce the memory requirements, but still offer higher resiliency than the original CNN, even with fewer parameters.

The second effect of E^{2}CNNs is the reduction of the activation buffers size. This reduction enables the additional robustness achieved with safe activations, at a smaller cost for their protection. This effect is a direct consequence of the pruning step, since removing an entire filter in a convolutional layer removes also the corresponding entire channel in the output feature map. Table 2 shows the percentage of memory words that each model needs to store the input data, the parameters and the two activation buffers to run each layer. The exact memory savings are shown in parenthesis. The increasing percentage of the memory footprint of the weights in the ensemble solutions corresponds to the decrease of the activations footprint (the absolute size of the weights remains almost constant). Deploying $M < N$ ensembles does not reduce the size required to store the activations, because we assume that they are executed sequentially, reusing the same memory space. These values confirm that an increasing number of instances in the E^{2}CNNs significantly reduces the size of the buffers, thus reducing the amount of memory that has to be protected.

5.3 Error-free accuracy of E^{2}CNNs

We first evaluate E^{2}CNNs with reliable memories (i.e., error-free accuracy tests), to compare its accuracy with the accuracy of the original (quantized) CNN. In Table 3, for each model, the highest achieved accuracy among the different evaluated solutions is highlighted in bold. We find that even if E^{2}CNNs is designed to improve error resiliency, it can also improve the accuracy of the original CNN in the absence of errors. This result is not the consequence of a higher number of parameters as it generally happens with SoA ensembles, because the number of parameters is similar in the initial CNN and in the corresponding E^{2}CNNs. Instead, the higher accuracy is the result of the prediction averaging used to build the ensemble: individually training each element of the ensemble makes the resulting models different from each other, even if they share the same structure. At inference time, this difference allows the final averaged predictions to be more accurate, because it takes into account the confidence of each member.

Additionally, we observe that increasing the cardinality of E^{2}CNNs initially results in higher accuracy, but it may lower it, for high values of N. This is more evident in LeNet-5, Industrial and AlexNet, and it is the consequence of the model compression applied: to obtain E^{2}CNNs with higher cardinality, the initial model size has to be pruned more and the resulting ensemble is not able to fully recover the accuracy drop. In particular, this happens when the original architecture is not largely oversized for the target dataset. In this situation, the pruning algorithm may not be able to preserve well the original accuracy, which may be critically reduced. An example is Industrial, for which we cannot build the 8-E^{2}CNNs, as the resulting compressed models reach accuracy levels lower than $50\,\%$.

5.4 Evaluation of E^{2}CNNs with SRAM errors

In this section, we explore the robustness of E^{2}CNNs against errors typical in SRAMs working at sub-nominal voltages (i.e., permanent bit errors). We conduct this analysis by comparing the accuracy of the original CNN with the accuracy of different E^{2}CNNs configurations. In all our experiments we store the activation buffers in a safe memory to better preserve the accuracy. We use the results obtained on the

TABLE 3
Accuracy comparison between the original (quantized) CNNs and 3 different E^{2}CNNs solutions (error-free evaluations)

Model	Original (%)	2-E^{2}CNNs (%)	4-E^{2}CNNs (%)	8-E^{2}CNNs (%)
LeNet-5	98.91	99.05	**99.06**	98,64
Industrial	98.29	**99.34**	94.87	-
AlexNet	85.20	**86.60**	85.80	81.60
GoogLeNet	99.70	99.70	**100.00**	99.90
MobileNet	95.125	95.75	**96.75**	95.37

TABLE 4
Energy consumption per access and bit error rate for an SRAM built on a $40\,\mathrm{nm}$ CMOS process at different voltage levels ($\mathrm{pJ/access}$) [14][33].

	Read	Write	Error Rate
850 mV	9.447	5.868	
750 mV	7.572	4.703	1×10^{-5}
700 mV	6.766	4.202	1×10^{-4}
650 mV	6.047	3.756	7×10^{-4}
600 mV	5.416	3.364	2×10^{-3}

four models evaluated in this section to build the heuristic presented in Section 3.3.

5.4.1 Robustness analysis

Voltage scaling reduces energy consumption in SRAMs, but it introduces errors in the memory array. Table 4 summarizes the read and write energy reductions achieved at sub-nominal voltages and, for each of them, the corresponding bit error probability [14], [33]. We test our benchmarks injecting either uniformly distributed errors or correlated errors, according to the computed error rates for different sub-nominal voltages.

Fig. 7 shows the accuracy of the original CNN and different E^{2}CNNs solutions at different error rates, both for uniformly distributed and correlated errors. On one side, these results demonstrate the improved robustness of the proposed E^{2}CNNs. We achieve up to $1.6\,\%$, $23.8\,\%$, $21.8\,\%$ and $7.0\,\%$ accuracy improvements in LeNet-5, Industrial, AlexNet and GoogLeNet, respectively. On the other side, we also observe that ensembles with more instances perform better at high error densities, while ensembles with less elements should be preferred in case of low error rates. This is clearly visible in Fig. 7c, where the 8-E^{2}CNNs performs worse than other ensembles with lower cardinality up to an error rate of 7×10^{-4}, but it reaches a better accuracy at the highest error rate. In Fig. 7b we can verify that there is a limit to the number of instances of the ensemble: the drop in accuracy observed for the 4-E^{2}CNNs in Table 3 makes this solution sub-optimal (i.e., the 2-E^{2}CNNs should be preferred across all the evaluated error rates). This highlights the importance of correctly selecting the cardinality of the ensemble. We find a similar behavior when we inject errors in the entire memory, hence not protecting the activation values.

Although on average the accuracy drop due to correlated errors is $1.1\,\%$ higher, with a maximum difference of $5.8\,\%$ at $600\,\mathrm{mV}$, the accuracy improvement of E^{2}CNNs is slightly higher in this case. These results (Figs. 7e–7h) show that E^{2}CNNs is applicable to both types of errors, even if correlated errors may initially have a stronger impact on CNN robustness.

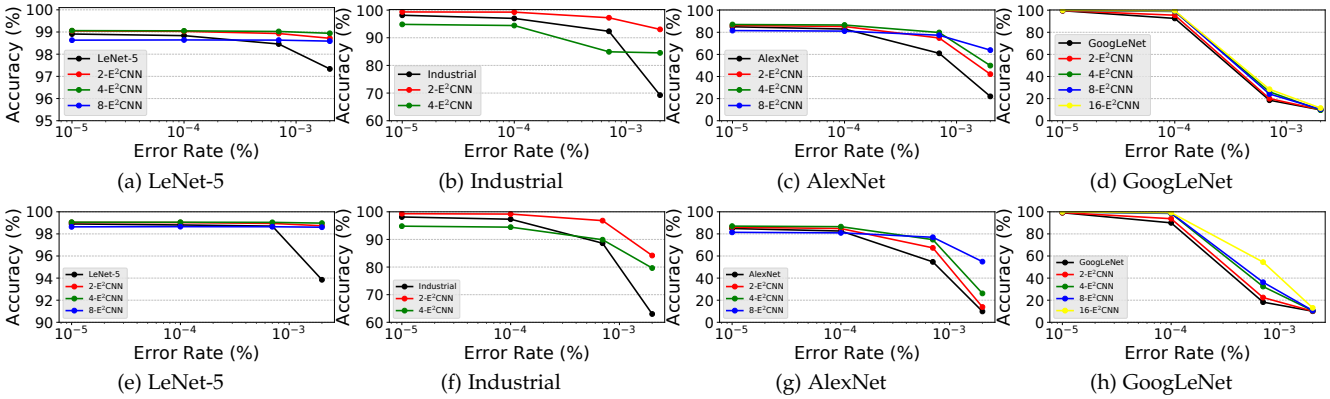
E^{2}CNNs increases the robustness of errors of the tested CNNs. More concretely, in AlexNet it is possible to drop the SRAM supply voltage by $150\,\mathrm{mV}$, which introduces an error rate of 1×10^{-4}, and still obtain better accuracy than with the original model at nominal voltage. In GoogLeNet, an equivalent error rate can be tolerated within $1\,\%$ of the original accuracy, but reduces the amount of memory that needs to be protected by $74\,\%$. Moreover, as we explore in Section 5.4.2, with E^{2}CNNs it is possible to run the 8-E^{2}CNNs for GoogLeNet with half the instances under the same error rate, and obtain a reduction in energy consumption of around $50\,\%$, all with only a $1\,\%$ of accuracy drop.

The previous results reflect the average system behavior for different error placements in memory, given a certain error rate. However, manufacturing industry and safety-critical applications are usually more interested in ensuring a target level of performance, given specific conditions. Knowing that some devices will exceed the average accuracy, while some others may be well below it, a yield analysis provides a more accurate estimation of the final outcome than average accuracy because it allows the designer to determine the proportion of the chips that can achieve a desired accuracy level for the expected error rate.

Fig. 8 shows the yield for a reference voltage of $650\,\mathrm{mV}$, which corresponds to a memory error rate of 7×10^{-4}. As a preliminary observation, we notice that E^{2}CNNs improves the yield for all the tested benchmarks. For example, in Fig. 8a only $90\,\%$ of the chips reach an accuracy level of $98\,\%$ in LeNet-5, while the entire chip population can reach this level when using the 8-E^{2}CNNs. Similarly, Fig. 8d shows that all the chips achieve an accuracy of $99\,\%$ when using the 16-E^{2}CNNs, while none of them could reach that level in the original GoogLeNet implementation; at a more conservative accuracy level of $95\,\%$, the (4,8,16)-E^{2}CNNs all have a yield of $100\,\%$, while the predicted yield for the original CNN is just $42\,\%$. Experiments on AlexNet (Fig. 8c) indicate an analogous behavior: $100\,\%$ of the chips can reach $79\,\%$ accuracy with 4-E^{2}CNNs, whereas no chip will reach it with the original model. Finally, in Fig. 8b we observe a yield improvement of $22\,\%$ for the percentage of chips that can reach $90\,\%$ accuracy when using the 2-E^{2}CNNs.

5.4.2 E^{2}CNNs as a mechanism for energy savings

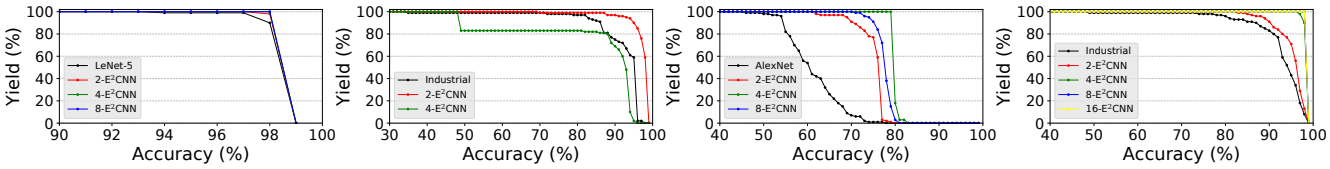
Here, we propose our analysis on how E^{2}CNNs can be used to save energy in edge devices equipped with SRAM. When operating these memories at sub-nominal voltages, the saved energy is the result of a reduced cost of memory accesses (Table 4). To estimate the energy reduction, we use our inference simulator to count the number of memory accesses. Then, we determine the consumed energy in the memory by multiplying the number of read and write operations by their individual energy cost. We do not take into account leakage power, nor the additional savings that may obtained in the processor, if operating at the same sub-nominal voltage. Therefore, the presented results can be considered as a lower bound. On average, we observe energy reductions up to $45\,\%$ when reducing



(a) LeNet-5 (b) Industrial (c) AlexNet (d) GoogLeNet

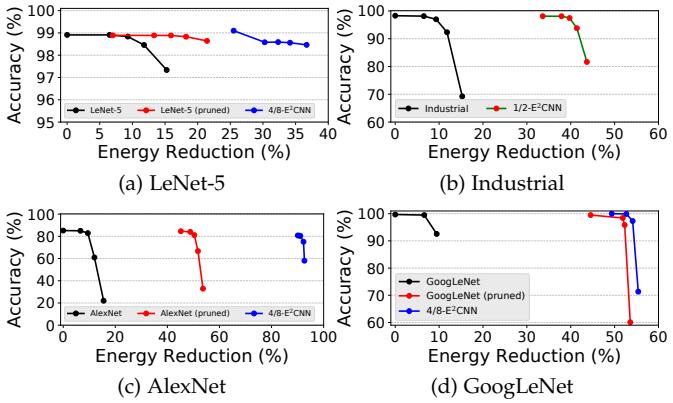
(e) LeNet-5 (f) Industrial (g) AlexNet (h) GoogLeNet

Fig. 7. Average accuracy for different error rates in the original CNN and E²CNNs using SRAM. Injected errors are uniformly distributed in (a–d) and with correlation along SRAM BLs in (e–h).



(a) LeNet-5 (b) Industrial (c) AlexNet (d) GoogLeNet

Fig. 8. Yield analysis on the original CNN and the E²CNNs at a memory error rate of 7×10^{-4} (1×10^{-4} in GoogLeNet, as all the configurations fail to achieve acceptable accuracy levels at higher error rates).



(a) LeNet-5 (b) Industrial

(c) AlexNet (d) GoogLeNet

Fig. 9. Average accuracy for different energy reductions in the original CNN and E²CNNs using SRAM. For each curve, the leftmost point corresponds to operation at nominal voltage ($850\,\mathrm{mV}$). Each point to the right corresponds to the next sub-nominal voltage point and error rate (Table 4). Energy reduction refers to the memory subsystem only.

the voltage from $850\,\mathrm{mV}$ (nominal) to $600\,\mathrm{mV}$ in our initial experiments. In particular, the achieved energy reduction obtained at a specific sub-nominal voltage varies among the tested benchmarks and also between the original CNN and proposed E²CNNs, as a consequence of a different number of MAC and memory operations.

Although voltage scaling is a well-known solution to reduce the energy consumption in SRAM, E²CNNs can be adopted as an orthogonal solution to obtain even better results. In fact, we have briefly discussed the possibility of deploying $M < N$ instances to build the ensemble. This approach reduces memory footprint, execution time and indeed energy cost. Compared to simply pruning the original CNN to obtain a smaller model, E²CNNs represents a better solution because its additional resiliency enables a more aggressive voltage scaling, with limited accuracy drops. Fig. 9 presents the accuracy level that can be reached for different amounts of energy reduction when using the original CNN (black), the pruned CNN having half of the initial memory requirements (red) and our E²CNNs in which only half of the expected number of instances are deployed (blue). In general, any value M of instances can be used for this purpose in a N-E²CNNs, with $1 < M < N$. However, our choice allows us to better demonstrate the benefits of E²CNNs by doing a comparison with the pruned version of the original model, having a similar memory footprint. The energy reductions in the original CNN are achieved solely through voltage scaling, while in the other two cases the obtained savings are the result of the synergic use of voltage scaling and model compression. Even if the two solutions share a similar memory size, the ensemble architecture achieves higher accuracies for the same energy saving, or, as an alternative, allows us to save more energy for the same accuracy level.

In particular, we observe in Fig. 9a that E²CNNs reduces energy consumption in comparison to the pruned CNN by a $10.3\,\%$, with an accuracy increase of $0.3\,\%$ in LeNet-5. In AlexNet (Fig. 9c), it reduces the energy cost by $50\,\%$ at a slight accuracy cost of $3.6\,\%$. In GoogLeNet (Fig. 9d), it jointly improves accuracy ($84.5\,\%$) and energy consumption (with a reduction of $55\,\%$), in comparison to the pruned CNN. Industrial (Fig. 9b) represents a peculiar case in this analysis: we know from Fig. 7b that the 2-E²CNNs is the optimal solution, because the dataset-specific design of Industrial makes the model compression algorithm unable to fully preserve the original accuracy, thus preventing high cardinality ensembles from recovering the initial precision.

This means that $1/2$-E^{2}CNNs corresponds exactly to the pruned version of Industrial. In this case, both solutions achieve an additional $24\,\%$ reduction in energy consumption, without affecting accuracy.

5.5 Evaluation of E^{2}CNNs with eDRAM errors

In the previous section we have presented our analysis on E^{2}CNNs applied to embedded systems equipped with SRAM. Here, we propose a similar analysis, but for devices using eDRAM. While only certain voltage levels can be applied in practice in SRAMs, the refresh period can be arbitrarily set in eDRAMs. Therefore, we can adopt an energy-driven approach to conduct these experiments. In this way, we compute in advance the required refresh rate to achieve a target energy saving in the memory subsystem.

We consider 4T GC-eDRAMs and a nominal refresh period of $18.7\,\mu s$ that ensures that more than $99.99\,\%$ of the memory cells do not fail [34]. Then, we relax the refresh intervals up to $90\,\mu s$, to investigate the trade-off between energy saving and accuracy drop.

In the particular case of CNN applications, we observe that model weights and biases are reused across consecutive inferences, as they are constant. That means that, after a short period, all the eDRAM cells that are vulnerable at the selected sub-nominal refresh rate will fail. In our experiments, we verified that this happens after the first layer of the first inference; in a sense, these errors acquire a static nature similar to the errors that appear in SRAMs. However, activation values are computed and written once per layer, then read in the next one; the time in between is the only period in which they are vulnerable. Therefore, at refresh rates close to the nominal (low error rates), and in the case of fast layers, activation values may not all be wrong before they are read. In that case, the exact error positions can be different for each inference. In a sense, the effective error rate in these cases is lower than the maximum error rate calculated for that refresh rate. Notwithstanding these considerations, for the case of eDRAMs we also observe significant accuracy drops when the activations are not protected (e.g., GoogLeNet, Industrial and AlexNet lose more than $25\,\%$ of accuracy at an error density of 1×10^{-4}). Therefore, we decided to conduct the rest of our experiments using safe activations. In the case of eDRAMs this is easier to control than with SRAMs because, rather than requiring independent voltage domains for each memory block, it is enough to adjust the refresh rate for each memory bank in the refresh controller.

5.5.1 Robustness analysis

To demonstrate the improved error tolerance, we first evaluate refresh period values that allow us to save a certain percentage of the total energy consumption of eDRAM, due to a reduction of memory accesses. These savings are computed for each benchmark, and range from $20\,\%$ up to $90\,\%$. Larger savings would result in error densities in the order of 1×10^{-1} and are not presented in this work, because none of the benchmarks can tolerate such high error densities. We observe a particular behavior for LeNet-5: the small size of this CNN makes the pruning unbalanced across its layers. Although the pruned models composing a desired E^{2}CNNs meet the desired memory constraint, the number of required MAC operations is not proportionally reduced. Therefore, the resulting E^{2}CNNs has a higher computational cost, which requires more aggressive refresh rate reductions to achieve the same energy savings as in the original CNN. The opposite effect can be observed for the 8-E^{2}CNNs of AlexNet, as this ensemble architecture has a disproportionately reduced number of MACs.

Fig. 10 presents the results of our experiments. It compares the accuracy of the initial CNN with that of our proposed E^{2}CNNs solutions. We consider protected activations in both the original model and E^{2}CNNs.[1] We observe that the 8-E^{2}CNNs can maintain an accuracy higher than $98\,\%$ at an error rate of 8×10^{-3}, for which the accuracy of the original model is largely reduced. We do not include the 4-E^{2}CNNs in Fig. 10a because its computational load requires critical refresh rate reductions to achieve the same energy savings as in LeNet-5, thus making this ensemble an impractical solution for real applications. In Industrial, the 2-E^{2}CNNs represents the optimal ensemble solution, while the high model compression required to build the 4-E^{2}CNNs prevents this solution to recover the initial accuracy. In Fig. 10b we observe that the 2-E^{2}CNNs has a higher accuracy than the original CNN at any error rate. A 4-E^{2}CNNs increases the accuracy up to $12.2\,\%$ in AlexNet (Fig. 10c) and up to $15.1\,\%$ in GoogLeNet (Fig. 10d). Interestingly, we notice again that E^{2}CNNs with higher cardinality perform better at high error rates, while E^{2}CNNs with less instances may be preferred at lower error densities.

5.5.2 E^{2}CNNs as mechanism for energy savings

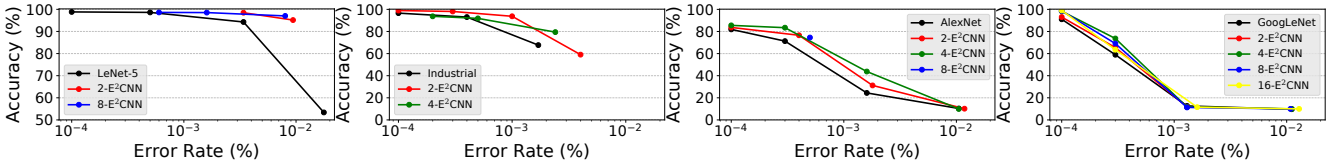
As was the case for SRAM, the best approach to save energy is the use of half E^{2}CNNs. Additionally, the refresh rate can be finely tuned to obtain additional savings. In this way, the energy savings stem from using a smaller model (less memory accesses) and from being able to use lower refresh rates while maintaining similar accuracy levels.

In Fig. 11, we compare the optimal E^{2}CNNs solution with only half of the instances against the pruned version of the initial CNN, which has similar memory requirements. These results show that in general E^{2}CNNs can better preserve the initial accuracy in presence of memory errors. As we discussed for Fig. 9b, the 2-E^{2}CNNs represents the optimal ensemble for Industrial and the corresponding half architecture coincides with the initial pruned CNN. However, for the other benchmarks our proposal outperforms the pruned CNN alternative: in Fig. 11a the $4/8$-E^{2}CNNs improves accuracy by $27.7\,\%$ when saving $60\,\%$ of energy. The $4/8$-E^{2}CNNs in Fig. 11c offers an accuracy still higher than $80\,\%$, achieving additional $30\,\%$ of energy reduction with respect to the pruned version and $90\,\%$ with respect to the original CNN. Finally, $14.5\,\%$ of accuracy improvement is obtained in Fig. 11d when saving $60\,\%$ of energy.

5.6 Evaluation of the E^{2}CNNs heuristic on MobileNet

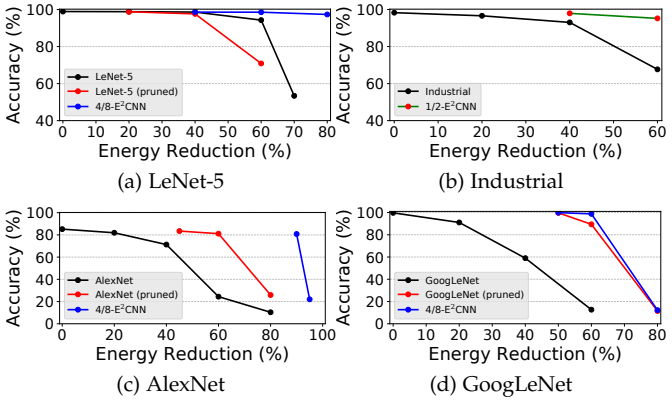
The heuristic method described in Section 3.3 is built upon the experimental results of LeNet-5, AlexNet, Industrial and

1. The points are not aligned along the x-axis because they correspond to the error rates associated with the refresh rates used in the energy-driven approach presented in Section 5.5.2: different architectures require different error rates to achieve a target energy reduction.



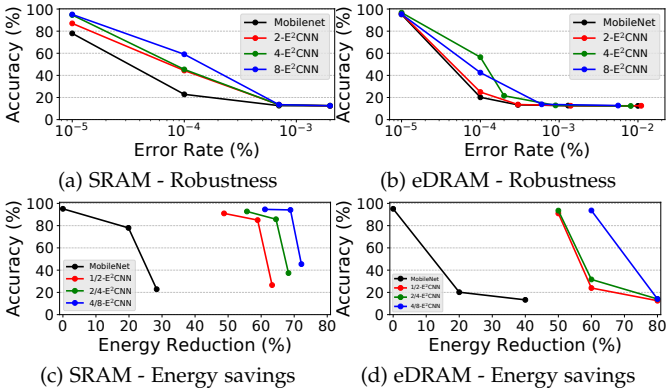
(a) LeNet-5 (b) Industrial (c) AlexNet (d) GoogLeNet

Fig. 10. Average accuracy for different error rates in the original CNN and E²CNNs using eDRAM.



(a) LeNet-5 (b) Industrial (c) AlexNet (d) GoogLeNet

Fig. 11. Average accuracy for different energy reductions in the original CNN and E²CNNs using eDRAM. For each curve, the leftmost point corresponds to operation at nominal refresh rate. Each point to the right, to the next sub-nominal refresh rate and associated error rate.



(a) SRAM - Robustness (b) eDRAM - Robustness (c) SRAM - Energy savings (d) eDRAM - Energy savings

Fig. 12. Average accuracy at different error rates and for different energy savings in SRAM (left) and eDRAM (right) for MobileNet.

GoogLeNet. Therefore, here we test it on MobileNet, to evaluate its effectiveness on a different CNN model.

For MobileNet, the heuristic selects the 4-E²CNNs architecture for high error rates, while the 2-E²CNNs is the architecture selected for lower error densities. Ensembles of higher cardinality are not considered as a pruned MobileNet, e.g., for 8-E²CNNs, has an accuracy drop larger than the maximum threshold used in the heuristic (5 %).

Table 3 showed that the 2-E²CNNs improves the original accuracy by 0.6 % in absence of memory errors. Then, Fig. 12a shows that the suggested solution offers additional robustness at an error rate of 1×10^{-5}, with an accuracy improvement of 9 %, while either the 2-E²CNNs or the 4-E²CNNs improve accuracy by 21 % at an error rate of 1×10^{-4} in SRAM. However, an exhaustive exploration reveals that the optimal solution is the 8-E²CNNs. In this case, the heuristic discarded that configuration due to the large accuracy drop in the individual instances, although the ensemble architecture is able to recover it—in contrast, ensembles based on AlexNet, Industrial and GoogLeNet cannot recover the initial accuracy level when pruning reduces the accuracy of their instances by more than 5 %. Our experiments suggest that this singular behavior might be related to the higher number of normalization layers present in MobileNet, but we were not able to train this architecture without these layers. Therefore, we can only propose this as a reasonable hypothesis for this behavior.

In Fig. 10d, we observe that all the proposed solutions maintain the initial accuracy for refresh rate reductions that produce error densities of 1×10^{-5}. However, the 4-E²CNNs suggested by the heuristic procedure offers the best accuracy improvement (+36.3 % at an error rate of 1×10^{-4}). Although the heuristic has not found the optimal ensemble, its conservative behavior prevents the selection of E²CNNs that cannot preserve the initial accuracy. Moreover, the predicted architectures can still be used to improve the error tolerance and achieve additional energy savings as shown in Fig. 12c and Fig. 12d. As example, we observe that the 2/4-E²CNNs can save 60 % of the total energy consumed in SRAM and 50 % of the total energy consumed in eDRAM, while maintaining an accuracy higher than 90 %.

6 Conclusions

In this paper we have introduced E²CNNs, a new design methodology to improve robustness against memory errors in embedded systems. In contrast to SoA ensemble methods, our proposed architectural solution can be deployed on constrained devices with no energy or memory overhead. Then, as part of our E²CNNs methodology, we have proposed a new a heuristic method that logarithmically checks possible architectures, selecting the most appropriate one for an expected error rate. We developed three error models, to simulate the behavior of SRAMs and eDRAMs operating at sub-nominal conditions. We have shown in our experiments that quantized (fixed-point) activations are more sensitive to memory errors than weights. Thus, activation buffers need to be consistently protected to better preserve the original accuracy, as we measured an average accuracy drop of 34 % when errors affect them. Moreover, E²CNNs decreases their size, reducing the amount of safe memory required. Our experiments show that E²CNNs improves error robustness for different error rates, in both SRAMs and eDRAMs. Furthermore, it enables additional energy savings, by reducing the number of instances used to build the ensemble. We demonstrated that E²CNNs represents a better alternative to CNN pruning alone because its increased robustness allows additional memory reduction and more aggressive energy savings, for the same accuracy level. Finally, we

found that relaxing the refresh rate in eDRAMs is more effective to reduce energy consumption than reducing the voltage in SRAMs because the error rates introduced for equivalent energy savings are lower, with a corresponding lower impact on accuracy.

REFERENCES

[1] M. Bojarski *et al.*, "End to end learning for self-driving cars," *arXiv:1604.07316*, 2016.

[2] F. Forooghifar *et al.*, "A self-aware epilepsy monitoring system for real-time epileptic seizure detection," *Mobile Networks and Applications*, pp. 1–14, 2019.

[3] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[4] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *J. Artif. Int. Res.*, vol. 11, pp. 169–198, 1999.

[5] T. G. Dietterich, "Ensemble methods in machine learning," in *Int. workshop on multiple classifier systems*. Springer, 2000, pp. 1–15.

[6] M. Amin-Naji, A. Aghagolzadeh, and M. Ezoji, "Ensemble of CNN for multi-focus image fusion," *Information Fusion*, vol. 51, pp. 201–214, 2019.

[7] L. Deng and J. C. Platt, "Ensemble deep learning for speech recognition," in *Conf. Int. Speech Communication Association*, 2014.

[8] S. Latifi, B. Zamirai, and S. Mahlke, "PolygraphMR: Enhancing the reliability and dependability of CNNs," in *Intl. Conf. on Dependable Systems and Networks (DSN)*. IEEE/IFIP, 2020, pp. 99–112.

[9] J.-H. Luo and J. Wu, "An entropy-based pruning method for CNN compression," *arXiv:1706.05791*, 2017.

[10] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, p. 32, 2017.

[11] A. Polyak and L. Wolf, "Channel-level acceleration of deep face representations," *IEEE Access*, vol. 3, pp. 2163–2175, 2015.

[12] J. Wu *et al.*, "Quantized convolutional neural networks for mobile devices," in *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.

[13] G. Li *et al.*, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Intl. Conf. for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[14] B. W. Denkinger *et al.*, "Impact of memory voltage scaling on accuracy and resilience of deep learning based edge devices," *IEEE Design & Test*, 2019.

[15] S. Koppula *et al.*, "EDEN: Enabling energy-efficient, high-performance deep neural network inference using approximate DRAM," in *Intl. Symp. on Microarchitecture*. IEEE/ACM, 2019, pp. 166–181.

[16] M. L. Seltzer, D. Yu, and Y. Wang, "An investigation of deep neural networks for noise robust speech recognition," in *Intl. Conf. on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 7398–7402.

[17] Y. Long, X. She, and S. Mukhopadhyay, "Design of reliable DNN accelerator with un-reliable ReRAM," in *DATE*. IEEE, 2019, pp. 1769–1774.

[18] C.-H. Cheng, G. Nührenberg, and H. Ruess, "Maximum resilience of artificial neural networks," in *Int. Symp. on Automated Technology for Verification and Analysis*. Springer, 2017, pp. 251–268.

[19] S. Han *et al.*, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[20] M. Widmer, A. Bonetti, and A. Burg, "FPGA-based emulation of embedded DRAMs for statistical error resilience evaluation of approximate computing systems," in *DAC*. IEEE, Jun. 2019.

[21] R. Giterman *et al.*, "An 800-MHz mixed-v_t 4T IFGC embedded DRAM in 28-nm CMOS bulk process for approximate storage applications," *Journal of Solid-State Circuits*, vol. 53, no. 7, pp. 2136–2148, 2018.

[22] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann.lecun.com/exdb/lenet*, vol. 20, p. 5, 2015.

[23] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[25] A. Krizhevsky, V. Nair, and G. Hinton, "The CIFAR-10 dataset," *online: http://www.cs.toronto.edu/kriz/cifar.html*, vol. 55, 2014.

[26] C. Szegedy *et al.*, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *AAAI Conf. on Artificial Intelligence*, 2017.

[27] P. Helber *et al.*, "Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification," *IEEE JSTARS*, vol. 12, no. 7, pp. 2217–2226, 2019.

[28] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[29] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[30] B. Reagen *et al.*, "Ares: A framework for quantifying the resilience of deep neural networks," in *DAC*, Jun. 2018.

[31] F. Frustaci *et al.*, "SRAM for error-tolerant applications with dynamic energy-quality management in 28 nm CMOS," *IEEE Journal of Solid-state circuits*, vol. 50, no. 5, pp. 1310–1323, 2015.

[32] M. Bauer *et al.*, "Semiconductor component with electromagnetic shielding device," Mar. 7 2006, US Patent 7,009,288.

[33] D. Bortolotti *et al.*, "Approximate compressed sensing: Ultra-low power biosignal processing via aggressive voltage scaling on a hybrid memory multi-core processor," in *Intl. Symp. on Low Power Electronics and Design (ISLPED)*. IEEE, 2014, pp. 45–50.

[34] R. Giterman *et al.*, "Current-based data-retention-time characterization of gain-cell embedded DRAMs across the design and variations space," *IEEE TCAS-I: Regular Papers*, vol. 67, no. 4, pp. 1207–1217, 2020.

Flavio Ponzina received the M.Sc. degree in Computer Engineering from Politecnico di Torino, Italy, in 2018. He is currently a PhD student at the Embedded Systems Laboratory (ESL), EPFL. His main research interests include low power architectures and AI-based systems optimization.

Miguel Peón-Quirós received the Ph.D. degree on Computer Architecture from the Complutense University of Madrid, Spain, in 2015. He is currently a postdoctoral researcher at EPFL. His research interests include energy and memory optimizations for embedded systems, and AI-enabled IoT devices.

Andreas Burg (S'97-M'05) received the Dr. sc. techn. degree from the Integrated Systems Laboratory of ETH Zurich, in 2006. In 2007 he co-founded Celestrius, an ETH-spinoff in the field of MIMO wireless communication. In January 2009, he joined ETH Zurich as SNF Assistant Professor. In January 2011, he joined EPFL, where he is leading the Telecommunications Circuits Laboratory. He was propoted to Associate Professor with Tenure in June 2018. He is currently an editor of the Springer Journal on Signal Processing Systems, the MDPI Journal on Low Power Electronics and its Applications, and of the IEEE Transactions on VLSI. He is also a member of the EURASIP SAT SPCN and of the IEEE TC-DISPS and the CAS-VSATC.

David Atienza is associate professor of electrical and computer engineering, and the director of the Embedded Systems Laboratory (ESL) of EPFL, Switzerland. He received an ERC Consolidator Grant in 2016, and was the recipient of the IEEE CEDA Early Career Award in 2013, and the ACM SIGDA Outstanding New Faculty Award (ONFA) in 2012. He is an IEEE Fellow and a Distinguished Member of ACM. His research interests include system-level design and thermal-aware optimization methodologies and ultra-low power edge computing architectures for wearable and IoT nodes.