Approximation-Based Fault Tolerance in Image Processing Applications

MATTEO BIASIELLI [ID]**, CRISTIANA BOLCHINI** [ID]**, (Senior Member, IEEE),**
LUCA CASSANO [ID]**, (Member, IEEE), ANDREA MAZZEO, AND ANTONIO MIELE** [ID]**, (Senior Member, IEEE)**

Matteo Biasielli is with King Digital Entertainment PLC., 11157 Stockholm, Sweden
Cristiana Bolchini, Luca Cassano, Andrea Mazzeo, and Antonio Miele are with the Dipartimento di Elettronica,
Informazione e Bioingegneria, Politecnico di Milano, 20133 Milano, Italy
This work was supported by Intel Corporation through Award titled Adaptive Application-oriented Fault Detection for Reliable Image Processing
CORRESPONDING AUTHOR: ANTONIO MIELE (antonio.miele@polimi.it)

ABSTRACT Image processing applications exhibit an intrinsic degree of fault tolerance due to i) the redundant nature of images, and ii) the possible ability of the consumers of the application output to effectively carry out their task even when it is slightly corrupted. In this application scenario the classical Duplication with Comparison (DWC) scheme, that rejects images (and requires re-executions) when the two replicas' outputs differ in a per-pixel comparison, may be over-conservative. In this article, we propose a novel lightweight fault tolerant scheme specifically tailored for image processing applications. The proposed scheme enhances the state-of-the-art by: i) improving the DWC scheme by replacing one of the two exact replicas with an approximated counterpart, and ii) allowing to distinguish between *usable* and *unusable* images instead of corrupted and uncorrupted ones by means of a Convolutional Neural Network-based checker. To tune the proposed scheme we introduce a specific design methodology that optimizes both execution time and fault detection capability of the hardened system. We report the results of the application of the proposed approach on two case studies; our proposal achieves an average execution time reduction larger than 30% w.r.t. the DWC with re-execution, and less than 4% misclassified unusable images.

INDEX TERMS Fault tolerance, image processing, approximate computing, convolutional neural networks, reliability

I. INTRODUCTION

IMAGE processing applications are largely employed for perception functionalities in unmanned-, autonomous-, or assisted-control in a variety of scenarios spanning from autonomous driving to robot and spaceship control. In such scenarios, computing systems are generally classified as safety-/mission-critical, and therefore, fault detection/tolerance mechanisms are generally applied to guarantee the necessary reliability level. State-of-the-art hardening solutions either employ special-purpose radiation-hardened devices [1], generally slower and more expensive than commercial ones, or rely on duplication/triplication and on a bit-wise comparison of the computed output [2], [3]. All these solutions entail a significant cost increase and performance degradation w.r.t. power consumption and execution time.

When exploring possibilities to reduce the overhead introduced by hardening techniques, two different considerations should be drawn. First, in systems belonging to the previously mentioned scenarios (e.g., automotive, robotic and aerospace) safety-critical applications and non-critical ones may co-exist. As an example, in an on-board satellite computing system, the navigation sub-system is a mission-critical component while the payload, frequently an image processing application, is not [4]. Moreover, image processing applications typically expose an inherent resilience to errors such that a certain degree of quality loss in the result can be intrinsically tolerated.

On the one hand, these applications work with input data that are frequently noisy or quantized, and this, in turn, introduces errors in the computations. On the other hand, the downstream user of the application output (either a human or another application) may be able to perform the subsequent tasks even if the computed output is noisy or partially altered. Such an intrinsic resilience has been exploited in the last decade under the Approximate Computing (AC) umbrella, trading-off the quality of the result with the performance/energy consumption of the system [5], [6].

In the context of fault tolerant computing for image processing applications, this intrinsic error resilience has been exploited in the recent past to define a new fault detection paradigm [7]. This paradigm shifts the focus from the classical identification of corrupted results based on a exact (bitwise) comparison, towards the classification of the results as *usable/unusable* based on the *prediction* of the capability of the downstream application to exploit them and to correctly perform the subsequent tasks. Hardening image processing applications based on this strategy allows for identifying cases where the corrupted output image does not affect the downstream application execution and final outcome. In these situations, any action to handle the occurrence of a problem that marginally altered the processed image would be redundant and should be avoided, as it introduces costs without providing real benefits. When considering software-based fault management techniques, it corresponds to the possibility of avoiding redundant re-executions causing time (and power) penalties.

The proposal in [7] exploits the *usable/unusable* paradigm and reduces the average recovery time, however it suffers from the same performance overhead as the Duplication with Comparison (DWC). Indeed, in a scenario where the two application's replicas and the checker are sequentially executed on a single-core CPU, the execution time is more than 2x w.r.t. the of the nominal application. To reduce this impact, AC has been exploited to reduce the replica computations in redundancy-based fault mitigation schemes, such as approximate/inexact Triple Modular Redundancy (TMR) [8]–[10], or fault detection ones [11], [12]. Common denominator to all these approaches is performing error checking/mitigation at the granularity of the single logic or scalar value, instead of considering the quality of the entire output image/signal as a whole to decide whether to trigger a re-execution or not.

Given these premises, we propose a lightweight fault tolerant scheme for image processing applications that extends the classical DWC. Our scheme combines application-level AC with the image usability classification paradigm defined in [7]; the former allows to drastically reduce the time overhead introduced by duplication (both when faults occur and when they do not) while the latter enables avoiding (when possible) unnecessary re-executions, to further save time (in the case of fault occurrence). We adopt a Convolutional Neural Network (CNN)-based *Smart Checker (SC)* instead of the classical Two-Rail Checker (TRC) to classify usable versus unusable images, as in [7]. On the other hand, differently from [7], we build the hardened system by pairing the nominal image processing pipeline with a number of lightweight *Control Blocks (CBs)*, each one dedicated to one of the stages of the pipeline. The CBs work on downscaled versions of the images taken in input by the corresponding pipeline stage: such application-level approximation allows the proposed scheme to save time w.r.t. both the DWC with re-execution and the solution in [7].

For the first time, we are here using the image downscaling technique, frequently used to trade-off results quality and power saving (as in [13]), for fault tolerance purposes. Such an approximation-based duplication approach leads to a novel, more challenging, definition of the CNN-based checker than in [7]. Indeed, the output images of the nominal replica and the ones produced by the CBs are different also when no faults occur during the processing. Therefore, the CNN-based checker must be able to analyse two images and differentiate *physiological* differences due to approximation, from *pathological* differences due to faults.

A preliminary version of the proposed approach has been presented in [14]; we here extend it to achieve a mature proposal that includes the following novel contributions:

- a refined and parametric architecture of the proposed fault tolerant scheme;
- a semi-automated design flow to configure the approximation level, and train and optimize the SC;
- a design-space exploration-based approach to tune the parameters of the CBs and SCs;
- an in-depth experimental campaign comprising two different image processing applications to demonstrate the effectiveness of both the proposed fault tolerant scheme and companion design flow.

We applied our proposal to two case studies: one for the identification of buildings in aerial pictures and the other one for motion detection in highway videos. The achieved time savings, measured on a single-core commercial embedded microprocessor, range from 36.52% to 33.39% w.r.t. the traditional DWC and [7]. As a counterpart to such benefits, the approach fails to correctly identify and handle corrupted unusable outputs with an incidence of less than 4%, that we believe is an acceptably low rate, comparable to [7].

The paper is organized as follows. Section II introduces the state-of-the-art related to fault detection/tolerance for image processing applications and approximate computing for fault detection/tolerance, our working context. Motivations and preliminaries of the proposed approach are presented in Section III, followed by our proposed fault tolerant scheme in Section IV and its companion design methodology (Section V). Section VI presents the considered case studies and the results of the experimental evaluation of the approach while Section VII draws conclusions.

II. RELATED WORK

Traditional hardening techniques introduce redundancies in the system to achieve fault detection or mitigation capabilities at the cost of a dramatic area or execution time increase. Approximate Computing (AC) has been exploited to limit such an overhead by lightening the redundant computations with the drawback of not offering a 100% fault coverage. Like for traditional hardening techniques, also AC-based techniques have been investigated and applied at different abstraction levels. We review here the most relevant ones.

Several approaches defined approximate hardening schemes at the logic level, by enhancing TMR (e.g., [8], [9], [15]), or DWC (e.g., [16], [17]) or focusing on the specific voter/checker component [18]. The focus is on a combinational function

having a single-bit output; the basic strategy these works adopt is to approximate the redundant circuit to minimize its area while limiting the error rate, i.e., the percentage of not detected/mitigated faults. Moreover, several logic synthesis approaches have been proposed to automatically apply such schemes [8], [9], [15].

When working at the Register-Transfer Level (RTL), a commonly-used AC technique to trade accuracy and circuit area is the reduction of the data precision of each processed scalar value. This technique has been adopted in [10], [19], [20], to define approximate TMR schemes where the redundant replicas elaborate only on a selected subset of the most significant bits. Therefore, a new voting module is defined to check the numeric distance between the fully-accurate nominal output value and the two lower-precision redundant ones; the effect is to confine errors in a subset of the least significant bits of the output values.

An alternative domain-aware approach for low-cost fault detection at the RTL consists in exploiting the specific peculiarities of the application under design to replace the exact replica with an estimator of the nominal functionality. Similarly to the previously presented reduced-precision redundancy approaches, the checker verifies whether the difference between the outputs of two redundant computations is lower than a given threshold. This idea has been first proposed in [11] for signal processing applications; it has also been demonstrated that for some specific applications the replica is even not necessary because the nominal output can be directly checked against the application input. The scheme is extended in [21] with the data precision reduction, and in [12] for fault mitigation adopting an N-Modular Redundancy (NMR) scheme.

The main difference between the existing approaches and the current proposal is that the former ones work on the single piece of processed information, being the single bit at logic level or a scalar value at RTL. Our proposal focuses instead on the entire processed information at once, i.e., the manipulated image, with the aim of deciding whether the output is usable by the downstream application, even if possibly corrupted, or not. All the discussed approaches can be adopted in the context of image processing to mitigate the effects of faults at the granularity of single pixels (or single bits). However, there is no straightforward extension to evaluating/checking the usability of the overall image. On the other hand, the idea in [11] of exploiting the specific peculiarities of the application domain to define approximate replicas is somehow similar to what we are here proposing.

There are very few works exploiting AC for fault detection/tolerance at system level. In [22] an approximation-based NMR scheme is defined at application level together with a mapping and scheduling approach; each software task is associated with a set of approximate replicas to reduce the execution time. Such a scheme requires an advanced voter module able to evaluate the similarity of redundant images processed at different approximation levels, however the paper does not discuss how this voter can be designed. A
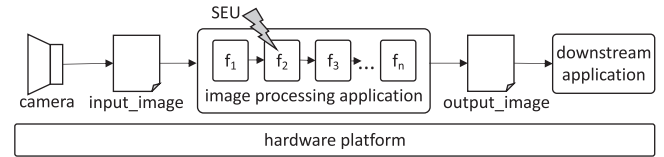

FIGURE 1. **Overview of the considered system.**

similar scheme is discussed in [23] where an application-level approximated DWC is proposed. The authors investigate the definition of a checker for comparing redundant results that are numerically different due to the approximation. The solution is similar to the one adopted for RTL schemes (such as [11]) based on the comparison of the numerical distance between redundant scalar results against a threshold. As a conclusion, this approach suffers from the same limitations of the previous ones if the aim is to evaluate the quality of the entire image.

As it has been discussed in the introduction, the only system level approach that analyses the usability of the entire output image is the one proposed in [7]. This approach exploits a CNN-based checker to perform a classification of the output image based on the usability of the output image. However, this approach suffers from the classical 2x overhead caused by the two exact replicas inherited by the DWC. The work proposed in the current paper moves from [7] and, based on the preliminary results presented [14], proposes a new solution, based on AC to evaluate the entire image as usable or not, and at the same time to drastically reduce the execution time of the hardened application. Finally, it is worth mentioning the system-level fault detection scheme proposed in [24] where, similarly to [11], no replica is required; however, this scheme is tailored to a specific image processing algorithm and cannot be applied in general.

III. PRELIMINARIES

We here present the working scenario and the necessary background to introduce the proposed approach.

A. THE APPLICATION MODEL AND HW PLATFORM

The proposed fault tolerance approach is meant for image processing applications that take in input images and produce filtered/enhanced images or a set of features extracted from the input image. As shown in Figure 1, the image processing application to be hardened is part of a larger system where the input images are taken from cameras or received from communication channels, and the output image/data is fed to a downstream application.

The application is internally composed of a pipeline of filters, each one receiving in input an image or a feature map (i.e., a multi-dimensional array) and producing a modified/enhanced image or feature map. We assume that the application to be hardened has already been designed and tuned to produce results at the specific quality level required by the designer. Therefore, we do not consider to apply approximation techniques to the application itself.

According to the literature, image processing applications can be executed on a variety of processing platforms. We here consider an embedded general purpose CPU commonly employed in mission-critical embedded systems, such as space missions (e.g., [25]). The execution model considers a hypervisor for the isolated execution of each filter in a time-triggered fashion, so that misbehaviors are not propagated [26]. Nonetheless, since the proposed approach works at the application level, it can be employed also on other processing platforms, such as GPUs and FPGAs; future work will explore these directions.

B. THE FAULT MODEL

In this paper we consider Single Event Upset (SEU) faults affecting CPU registers, assuming the cache and main memory to be protected via Error Correction Codes (ECC). Moreover, we assume the input image and the CNNs parameters to be always loaded from mass storage devices, which, again, are typically protected via ECC. Provided that faults are rare events, we here assume that a single SEU may occur during one run of the application. When considering the execution of the application within the hypervisor, the single SEU may cause the following effects in the execution of the single filter: i) software crashes or operating system/hypervisor exceptions blocking the execution, ii) software hangs leading to a non-termination, and iii) Silent Data Corruptions (SDCs), where the executed filter terminates producing an erroneous output, without any alert [27]. It is safe to assume that faults causing effects in the first two classes are autonomously detected by the hypervisor by using, for example, watchdogs. On the other hand, SDCs represent the most critical class of effects, requiring the adoption of fault detection/tolerance mechanisms, focus of our work.

C. IMAGE USABILITY CLASSIFICATION

We borrow the concept of *usability* proposed in [7] that may be summarized as follows. Considering the system presented in Figure 1, the (intermediate) result of a filter within the image processing application affected by a fault during its execution is classified as *usable* if the corresponding final output of the application will lead the downstream counterpart to take the same decision that would have been taken when using a fault-free output, otherwise it is *unusable*.

As an example, let us consider the case of a motion detection application[1] employed to identify cars moving on a highway, and a downstream application building a 3D model of the scene to subsequently compute a trajectory for an autonomous driving car. Figures 2(a), 2(b) and 2(c) show the input image, the intermediate image showing the areas where a motion is detected and the output image enhanced with bounding boxes around the identified moving cars, respectively. In case an SEU occurs during the processing of a filter inducing an observable alteration of the final output image, one of two following situations apply:
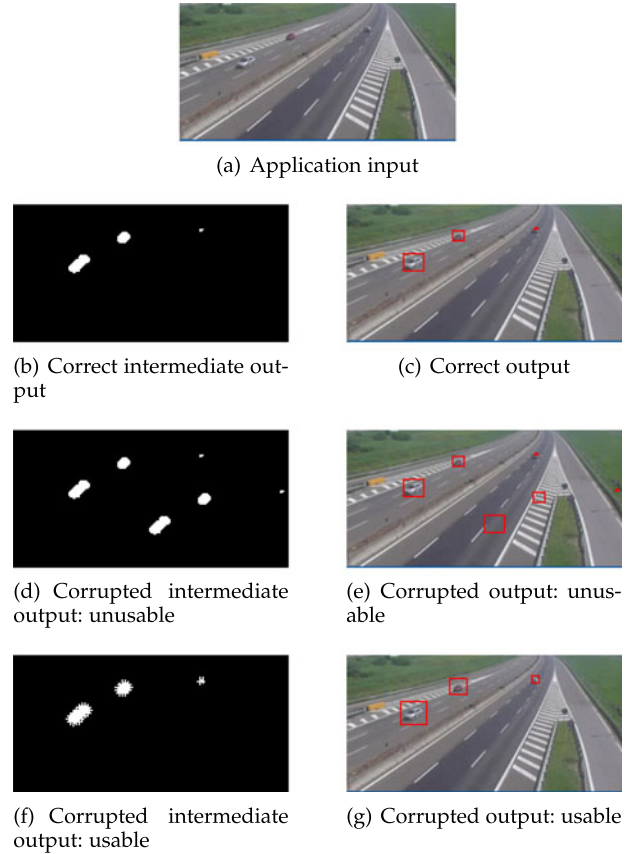
[1]All details of this case study will be provided in Section VI.

(a) Application input

(b) Correct intermediate output

(c) Correct output

(d) Corrupted intermediate output: unusable

(e) Corrupted output: unusable

(f) Corrupted intermediate output: usable

(g) Corrupted output: usable

FIGURE 2. Examples of fault effects on processed images.

i) The fault causes a relevant alteration in the detection of the motions (Figure 2(d)), such that three wrong bounding boxes are added (Figure 2(e)). Thus, the downstream application will add to the 3D model three "ghost" cars introducing wrong obstacles in the trajectory computation. The corrupted output is therefore unusable.

ii) The fault causes a minor alteration consisting in larger blobs in the intermediate image (Figure 2(f)); the effect in the final image (Figure 2(g)) consists in bounding boxes with a slightly different shape. In such a situation, the downstream application will decide correctly the trajectory therefore the corrupted output is usable.

While the classical difference between correct and corrupted output is general, the classification of the output w.r.t. the usability is strictly context-related. Moreover, like in [7], usability classification is then employed for fault tolerance purposes. A fault tolerant scheme based on such estimated usability analysis may decide a possibly corrupted intermediate output image is to be discarded (D) being considered too damaged to be usable, or not discarded (/D). Then, to evaluate the correctness of such a decision, we may ideally use such a corrupted output image to feed the downstream application so that we have the actual response, i.e., usable (U) or unusable (/U). Table 1 reports the resulting four classes. In particular, due to the stochastic nature of the CNN classification leading to D versus /D, this fault tolerant scheme may incur in errors: i) a usable image is discarded (false positive,

TABLE 1. Performance evaluation: notation and classification.

D	Discarded	the scheme classifies the output as unusable
/D	Not Discarded	the scheme classifies the output as usable
U	Usable	the output is usable
/U	Unusable	the output is unusable
D /U	Correctly handled corrupted output	
/DU	Achieved execution time savings	
DU	Not exploited execution time savings – false positive	
/D /U	Erroneously accepted data – false negative	

i.e., D U), or ii) an unusable image is accepted (false negative, i.e., /D /U). False positives affect the efficiency of the fault tolerance approach in terms of execution time increase, because re-computations are performed when not needed. On the other hand, false negatives affect the effectiveness of the solution, impacting the achieved reliability level.

D. THE BASELINE SOLUTIONS

A baseline approach for fault tolerance in the considered scenario is represented by DWC with re-execution: two exact replicas of each filter composing the application are executed in sequence and a routine implementing a TRC is run to perform a bit-wise checking on the replicas' outputs. Upon detection of an error the filter is re-executed. The bit-wise comparison triggers a re-execution independently of the impact of the corruption on the processed image, thus being a very conservative solution. We do not compare against the TMR solution typically adopted to achieve fault tolerance, because it is more expensive than the DWC with re-execution, introducing an additional replica also in fault-free executions (having the voter and TRC similar overheads). We also refer to another baseline solution, the scheme proposed in [7], where usability analysis is exploited although using a nominal replica of the application.

The metrics we adopted to evaluate our proposal and to compare it against the baseline solutions are: i) *the average execution time* of the single run of the application to measure the overhead introduce by fault tolerance, and ii) *the percentage of fault negatives* (/D /U), to measure the actual vulnerability of the hardened solution.

IV. THE FAULT TOLERANT SCHEME

Figure 3 depicts a block-diagram view of the proposed fault tolerant scheme applied to a generic image processing pipeline. Each filter f_i is paired with a Control Block (CB) cb_i that is
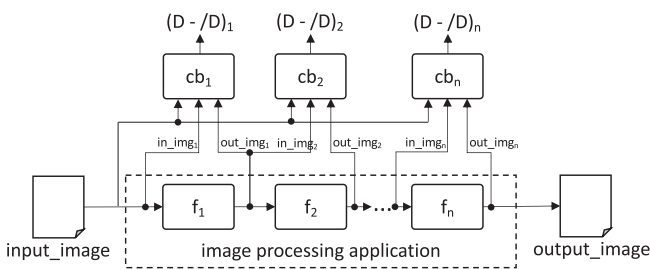

FIGURE 3. A high-level representation of the proposed fault tolerant scheme.
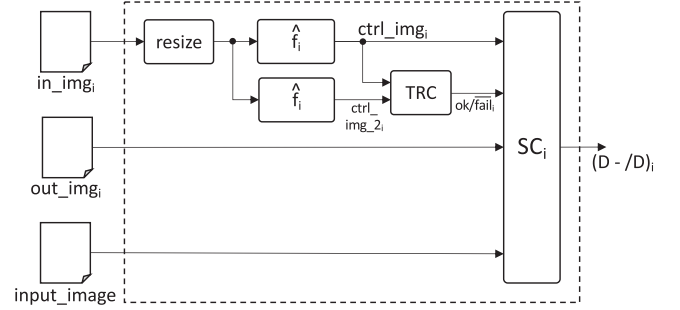

FIGURE 4. Internal structure of cb_i.

used to decide whether to discard (D) the output of f_i or not (/D). As we previously mentioned, our proposal adopts the image usability classification paradigm. Therefore, the goal of each CB is to decide whether to discard the possibly corrupted output image of the corresponding filter by *predicting* the usability of the final output image (*output_image*) that the pipeline will produce. To obtain such usability prediction each CB is equipped with a Smart Checker (SC) that features a CNN. The CNN has to distinguish between i) physiological differences between the nominal (intermediate) output and the approximated output due to approximation and ii) pathological differences due to the occurrence of faults during the execution. Indeed, approximation and faults cause dissimilar distributions in the pixel-wise difference between the nominal and the approximated outputs; the CNN has to learn these differences to correctly classify usable versus unusable cases.

It is worth mentioning that, after discarding the output of a filter, the proposed scheme adopts re-execution at the granularity of the single corrupted filter to compute the correct result. Moreover, as a general consideration, we point out that in our proposal, replication has to be intended as time redundancy although, for the sake of clarity, it is represented as space redundancy in the figures.

In the remainder of this section we will present the internals of the CB and the SC.

A. THE CONTROL BLOCK ARCHITECTURE

The internal structure of the generic CB cb_i associated with filter f_i is depicted in Figure 4. It is composed of an *input downscale module*, two replicas of f_i working in parallel, namely $\hat{f}_i$, a TRC and the SC. The generic cb_i receives the input and output images of f_i, namely *in_img$_i$* and *out_img$_i$*, and *input_image*, i.e., the global input image of the pipeline. Image *in_img$_i$* is first downscaled by a factor d_i (applied to both image dimensions) and then the two replicated filters $\hat{f}_i$ are executed in parallel on the downscaled image. The execution on downscaled images constitutes the application-level approximate computing technique exploited by the proposed scheme to achieve lightweight fault tolerance.[2] The output

---

[2]It is worth pointing out that the structure and working principles of the proposed fault tolerant scheme are independent of the adopted AC technique. However, the specific parameters, e.g., CNN weights, downscaling factors, are designed and optimized based on the adopted AC technique, as well as on the considered application.

images of the two replicated filters, namely $ctrl\_img_i$ and $ctrl\_img\_2_i$, are bit-wise compared with a classical TRC and the response, dubbed $(ok/\overline{fail})_i$ is fed to the SC. The SC takes in input also $ctrl\_img_i$, out_img_i and $input\_image$.

The downscaling factor d_i is a key parameter to optimize the execution time of the hardened application and the fault detection accuracy at the same time. The larger d_i, the smaller the image processed by the redundant replicas and consequently the shorter the execution time. On the other hand, a large d_i may negatively affect the classification accuracy of the SC since the resulting $ctrl\_img_i$ and $ctrl\_img\_2_i$ images may be too small and lose relevant details. Nonetheless, the largest value for d_i highly depends on the sensitivity and the complexity of each specific filter in the application pipeline. Therefore, to minimize the execution time of the hardened application and to maximize the fault tolerance capability, d_i needs to be specifically optimized for each CB, as described in Section V.

The DWC scheme between the redundant replicas in the CB allows for detecting faults corrupting either $ctrl\_img_i$ or $ctrl\_img\_2_i$. In the case a single filter is instantiated in the CB, a fault corrupting $ctrl\_img_i$ or $ctrl\_img\_2_i$ may induce the SC to classify out_img_i as unusable although it is actually uncorrupted. On the other hand, the DWC schema in the CB detects and notifies the occurrence of any fault affecting either $ctrl\_img_i$ or $ctrl\_img\_2_i$ on the $(ok/\overline{fail})_i$ output. Therefore, under the adopted single fault assumption (as it will be discussed in the following subsection) the SC determines the output of the nominal filter not to be corrupted when the $(ok/\overline{fail})_i$ output is `fail`, preventing the CNN execution. When $(ok/\overline{fail})_i$ is `ok` no assumptions can be made by the SC and the CNN needs to be executed.

Finally it is worth mentioning that $ctrl\_img_i$ and $ctrl\_img\_2_i$ are exclusively exploited for fault tolerance purposes, and they can never be considered as an alternative output of the nominal pipeline due to their sensibly lower quality w.r.t. the nominal counterpart. Therefore, even if the proposed scheme comprises three modules for each filter (the nominal filter and the two replicated ones), the solution does not represent an approximate TMR as done in [10], where the system tolerates the occurrence of faults and is able to provide an approximated result in case the nominal pipeline fails.

B. THE SMART CHECKER ARCHITECTURE

This is the main component of the proposed fault tolerant scheme, constituting the *brain*, in charge of deciding whether to discard images and trigger re-executions (`D`) or not (`/D`). As previously discussed, one SC is instantiated for each CB. A high-level representation of the SC for a generic nominal filter f_i is shown in Figure 5. The ith SC receives i) the output image of f_i (out_img_i), ii) the output image produced by one of the two $\hat{f}_i$ ($ctrl\_img_i$), iii) the input image of the pipeline ($input\_image$) and iv) the $(ok/\overline{fail})_i$ message produced by the TRC. Based on these three images the SC decides whether to discard out_img_i and to trigger a re-execution of the ith stage of the pipeline or not. The reason for the SC to be fed also
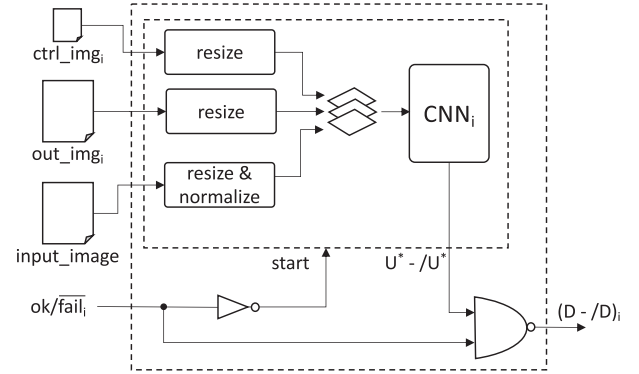

FIGURE 5. Internal structure of the **i**th SC.

with the $input\_image$ is that the comparison between out_img_i and $ctrl\_img_i$ only allows to identify the modification introduced by a fault in the image manipulated by the nominal processing. However, to predict the impact of such modification on the usability of the final output we empirically observed the need for correlating it with the $input\_image$.

Since out_img_i and $ctrl\_img_i$ are not identical also in a fault-free execution, the SC must discriminate between differences due to approximation and those caused by the occurrence of faults (on top of approximation). Furthermore, when the latter situation occurs, the SC must predict the impact on the final output, such that it will be usable or not, and in this case the output of the ith stage of the pipeline is discarded and a filter re-execution is triggered. To do so, each checker exploits a CNN that will notify the predicted usability (U^*) or not ($/U^*$).

To summarize, when the $(ok/\overline{fail})_i$ signal is a `fail`, the output of the SC will be automatically set to $/D$: in this scenario a fault indeed occurred in one of the two replicated filters, therefore the output of the nominal one is fault-free. When the $(ok/\overline{fail})_i$ signal is `ok`, the CNN is invoked. As mentioned and shown in Figure 5, images img_i, $ctrl\_img_i$ and $input\_image$ are downscaled before being fed to the CNN. In fact img_i and $input\_image$ are larger than $ctrl\_img_i$, while the designed CNN input is required to be a single three-dimensional matrix, where the three images are stacked. Therefore, img_i and $input\_image$ are downscaled to match the size of $ctrl\_img_i$. Moreover, as it has already been demonstrated in [7], images img_i and $input\_image$ can be sensibly downscaled to reduce the execution time of the CNN while preserving its classification accuracy. Indeed, the redundant filters are more susceptible to aggressive downscaling than the CNN. Therefore, img_i and $input\_image$ can be additionally resized with a d_{cnn_i} factor, and $ctrl\_img_i$ consistently with a $\frac{d_{cnn\_i}}{d_i}$ downscaling factor, thus obtaining three smaller images of the same size to be stacked. Finally, should $ctrl\_img_i$ and img_i be feature maps or heat maps (instead of images) having values in a range different from the color intensity of the pixels of $input\_image$, $input\_image$ is normalized. This is a necessary step to create more uniform inputs and ease the CNN training, also increasing the model prediction capability.
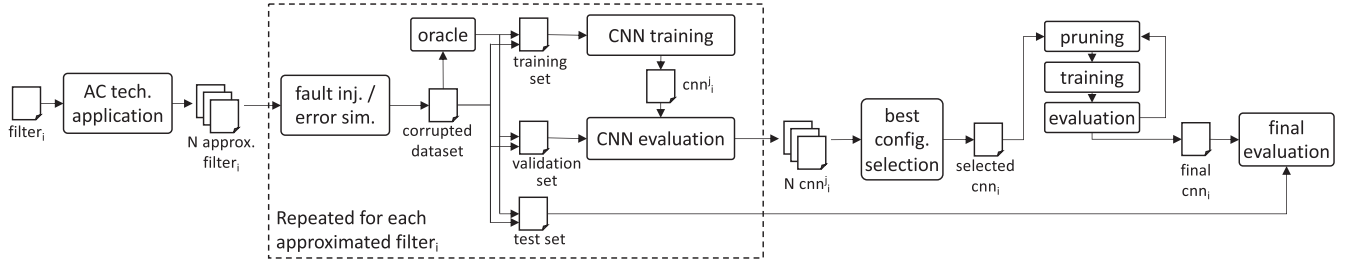
FIGURE 6. The design flow for cb_i and related SC.

The output of the CNN is a probability value u_{cnn} that estimates the usability of the final output of the application; the value is finally compared with a threshold U_{th}, producing the final classification, U^* or $/U^*$.

V. THE SMART CHECKER DESIGN METHODOLOGY

The design of the proposed fault tolerant scheme involves several aspects that need be tuned opportunely to achieve the desired accuracy and time savings, and because each stage of the pipeline consists of a filter with its own peculiarities, the associated CB is independently designed. More precisely, we defined a methodology that, for each ith CB, supports the implementation of 1) the AC technique to be adopted for the approximated filter replicas (and the preceding resize module), 2) the resize and normalization blocks in the SC, and 3) the CNN in the SC.

The design flow, shown in Figure 6, starts by analyzing the specific i^{th} filter f_i and by selecting a number of approximations. The outcome of this first activity is a list of approximated versions of f_i, namely $\hat{f}_i^j$. For each $\hat{f}_i^j$, a CB, and the corresponding SC, is designed; to this end, the necessary training, validation and test datasets have to be defined. The procedure is based on the generation of a large set of corrupted images by applying classical fault injection (or error simulation) techniques. The corrupted images are labelled as U or $/U$ by means of an *Oracle*. Indeed, the Oracle, borrowed from [7], mimics the downstream application, thus it is able to predict whether a corrupted output of the image processing application will be usable. Such sets are then used to run the classical training and evaluation procedures for the CNN within the CB.

Given a filter f_i of the nominal application, the design of an SC for each approximated filter $\hat{f}_i^j$ produces a solution space where each point presents a different trade-off w.r.t. the two considered metrics: i) the percentage of unusable images erroneously accepted by the SC (false negatives), and ii) the average execution time of the protected application. The former is the most critical one from a reliability point of view, impacting the accuracy of the hardening strategy, whereas the latter measures the benefits of the solution. Moreover, for each CB and the included CNN, the trade-off between the aforementioned metrics can still be tuned by leveraging the U_{th} threshold. Indeed, U_{th} may be interpreted as the degree of confidence on the usability of the inputs the CNN requires for the image to be classified as usable. Intuitively, the larger U_{th},

the less the images that will be accepted by the CNN. This, in turn, has the potential to reduce the amount of false negatives (unusable corrupted images wrongly accepted) but increase the amount of false positives (usable corrupted images or even uncorrupted images that lead the CNN to trigger unnecessary re-executions). Given the solution space including the explored alternatives, we select the CNN with the best trade-off between the percentage of false negatives and the time saving, to optimize its implementation by means of pruning.

When all stages are finalized, they are integrated to build the overall hardened application, for an assessment of the resulting architecture w.r.t. accuracy and time savings.

The rest of the section details the various steps of the design flow; all steps are applied to all i stages of the application pipeline, before the final integration.

A. APPROXIMATE COMPUTING TECHNIQUE APPLICATION

The starting step of the design flow is the selection of the appropriate AC techniques to generate approximate filters $\hat{f}_i^j$ for the filter f_i. The designer is the one to perform the selection based on the expertise and knowledge on the entire application. Although the methodology is general and various AC techniques can be exploited, for the sake of simplicity in this work we adopt the *downscaling* technique and the discussion refers to it. In this context, the designer selects a set of suitable values for the downscaling factor d_i, being 1 the lower bound, that corresponds to no approximation/downscaling. The upper bound d_{max_i} is identified by adopting the strategy in [7]. More precisely, to identify such an upper bound we iteratively increase the downscaling factor of the inputs of the CNN, while not approximating the replica, until the obtained classification accuracy is high and stable before a change of trend. Once d_{max_i} is identified, values in the $[1, d_{max\_i}]$ interval are included by selecting all possible divisors of the sizes of the input image.

B. TRAINING, VALIDATION AND TEST SETS GENERATION

Each CNN requires three sets of samples: the training, validation and test sets, each one including both pristine and corrupted images, that constitute – for each set – the *fault-free subset* (e.g., $TRS_{/faulty}$, $VS_{/faulty}$ and $TES_{/faulty}$) and the *faulty subset* (e.g., TRS_{faulty}, VS_{faulty} and TES_{faulty}), respectively. Since all procedures for the generation of these sets are

similar, we present the process for the training set generation, leaving the application to the validation and test sets to the reader.

The *fault-free training subset TRS$_{/faulty}$* is generated by processing a set of images in nominal conditions, i.e., without any fault. Input images should be representative of the working scenario where the pipeline will work, e.g., similar altitude, range of colors, sizes. For a nominal filter f_i and the corresponding approximate filter $\hat{f}_i^j$, each item in the training set is a tuple of the form:

$$< input\_image, out\_img_i, ctrl\_img_i^j, output\_image, \texttt{U} >,$$

where, according to Section IV, *input_image* and *output_image* represent the global input and output of the application, respectively, *out_img$_i$* is the output of f_i and *ctrl_img$_i^j$* is the output of $\hat{f}_i^j$. *output_image* is the pristine output of the nominal processing, therefore the tuple is labelled as $\texttt{U}$.

The *faulty training subset TRS$_{faulty}$* is generated by corrupting the execution of filter f_i. Each item in the set is a tuple of the form:

$$< input\_image, out\_img_i^*, ctrl\_img_i^j, output\_image^*, u >.$$

The input of this filter is the nominal output produced by the previous stages of the pipeline, while the execution of filter f_i is corrupted because of the fault effect, having an impact on the computed output *out_img$_i^*$*. This intermediate altered output is propagated to the remaining stages of the pipeline to produce the final global potentially modified output *output_image**. The usability of *output_image** is determined by feeding this image as input to the *Oracle*, which in turn decides whether the downstream application would correctly carry out its task using the image or not. Such usability prediction is used to label the tuple, *u*, as usable ($\texttt{U}$) or not ($\texttt{/U}$). That is to say, we train each checker to "predict" the usability of the final corrupted pipeline's output image based not only on how the fault modified the image produced by the specific stage affected by the fault but also on how the functionality implemented by the subsequent stages of the pipeline is affected to the corrupted image.

The corruption of the execution of an application is generally carried out by means of the classical fault injection. In our scenario the generation of the faulty training subset represents a relevant issue: indeed, it is necessary to generate a very large number of intermediate images *img$_i^*$* really corrupted by the fault. However, when running a fault injection campaign, it is common that many of the experiments lead to uncorrupted results because either the fault is not activated or it is absorbed by the subsequent elaborations. Therefore, we here adopt the strategy proposed in [7], where the generation of corrupted images is split into two subsequent steps. First, a fault injector specific for the considered hardware architecture is used to inject faults in the system executing filter f_i and to collect a set of corrupted images. Then, by analysing these corrupted results, recurrent *error models* can be identified and then used

in an error simulation campaign. This alternative solution exploiting a *tailored* error simulation allows for a faster generation of a large large set of corrupted intermediate outputs (*img$_i^*$*) and associated global outputs (*output_image**).

C. CNN TRAINING AND EVALUATION
A standard approach based on the Adam optimizer [28] is adopted to train the CNN by means of the training set defined in the previous step of the design flow. Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions that has proven to work well in several Neural Network optimization problems. It combines the benefits of AdaGrad [29], an optimization algorithm performing well with sparse gradients, and RMSProp [30], suitable in on-line settings. Since the training problem is quite similar to the one presented in [7], we tuned the process, its parameters and objective loss function in the same way. Moreover, to prevent overfitting, Early Stopping [31], Dropout [32] and Ridge Regularization [33] are employed.

When the training is completed, the CNN is evaluated by means of the validation set; in particular, for each tuple of the validation set the CNN is fed with *input_image*, *out_img$_i$* (or *out_img$_i^*$*) and *ctrl_img$_i$* contained in each tuple; the corresponding CNN output value u_{cnn} is computed and it is appended to the tuple itself.

D. BEST CONFIGURATION SELECTION
The result of the previous steps is a set of CNNs, one for each $\hat{f}_i^j$. Each CNN is still parametric since the threshold U_{th} (which the output of the CNN, u_{cnn}, is compared against) has not yet been specified. Moreover, only after a value is set for U_{th}, the classification accuracy of the obtained CNN (and thus also of the SC) can be evaluated. When considering the fault-free validation set $VS_{/faulty}$, since all images are pristine, $\texttt{D /U}$ and $\texttt{/D /U}$ are empty and any variation of U_{th} will only affect performance.

When considering the *faulty validation set VS$_{faulty}$*, the selection of value $U_{th} \in (0, 1)$ drives the trade-off between accuracy and time savings of the final solution. By moving U_{th} towards the upper bound, accuracy is favored as the number corrupted images that will trigger a re-execution increases (by increasing the number of $\texttt{D /U}$ and $\texttt{D U}$ cases). By moving U_{th} towards the lower bound, the opposite effect is achieved, by increasing the number of $\texttt{/D /U}$ and $\texttt{/D U}$ cases, corresponding to lower accuracy and higher time savings.

As a result, the solution space we explore is defined by i) all CNNs associated with the possible approximated filters $\hat{f}_i^j$ ii) for each CNN, all possible values $U_{th} \in (0, 1)$. Each combination of these elements identifies an SC implementation, evaluated and compared to identify the most promising solution.

A multi-objective selection approach needs to be adopted to optimize two conflicting aspects. On the one hand, the CNN is required not to miss the identification of unusable corrupted images; this first optimization metric is computed as the ratio between the number of tuples classified $\texttt{/D /U}$ in VS_{faulty} and the overall size of such a subset:

$$\%/\text{D}/\text{U} = \frac{|/\text{D}/\text{U} \in VS_{faulty}|}{|VS_{faulty}|}. \tag{1}$$

On the other hand, the overall hardened application has to exhibit high performance; thus, the second metric is the average execution time of the filter, computed as follows:

$$T_{exec} = T_f + 2 \cdot T_{\hat{f}} + (T_{cnn} + T_{ds_{in}} + T_{ds_{out}}) + T_f \cdot \%\text{D}_{VS}, \tag{2}$$

where the overall time is computed by summing the execution times of the nominal filter T_f, of the two replicated filters $T_{\hat{f}}$, of the checker (expanded in the execution times of the downscaling steps $T_{ds_{in}}$ and $T_{ds_{out}}$ and of the CNN T_{cnn}). The re-execution time is computed as T_f weighted by the percentage of times an image is discarded $\%\text{D}_{VS}$.

The estimated percentage value of re-executions for both the fault-free subset ($\%\text{D}_{VS_{/faulty}}$) and the faulty one ($\%\text{D}_{VS_{faulty}}$) within the validation set can be computed as the ratio between the number of discarded images D (D /U and D U) and the overall size of the subset. We here report the formula for the faulty validation subset VS_{faulty}:

$$\%\text{D}_{VS_{faulty}} = \frac{|\text{D}/\text{U} \in VS_{faulty}| + |\text{D}\text{U} \in VS_{faulty}|}{|VS_{faulty}|}. \tag{3}$$

Do note that $\%\text{D}_{VS_{/faulty}}$ can be computed in the same way; no faults have been injected, therefore the number of D /U is 0. Then, the two results can be combined based on the expected fault rate f_{rate} of the working scenario:

$$\%re\text{-}ex = \frac{\%\text{D}_{VS_{free}} + f_{rate} \cdot \%\text{D}_{VS_{faulty}}}{1 + f_{rate}}. \tag{4}$$

It is worth mentioning that in a common scenarios f_{rate} is generally very small, therefore, most of the time is spent in the fault free scenario. As a result, $\%\text{D}_{VS}$ is almost the same as $\%\text{D}_{VS_{free}}$. This formulation allows us to highlight the significant reduction of the time overhead of the adopted fault tolerance mechanism with respect of existing solutions, and in particular [7].

Each CNN (i.e., each filter) is therefore characterised by two indices, $\%/\text{D}/\text{U}$ and T_{exec}, associated with the setting of its U_{th}. We plot on a curve all the points corresponding to the different possible values of U_{th}, in the $\%/\text{D}/\text{U}$ and T_{exec} space, obtaining a Pareto front.

The result of this evaluation is a set of curves, one for each CNN, in a chart presenting $\%/\text{D}/\text{U}$ and T_{exec} on the two axes. We get a (large) set of points, each one corresponding to a d, U_{th} pair, as shown in Figure 7, where we use different colors for different d values, for additional information.

When the distribution of the points is evenly distributed w.r.t. the $\%/\text{D}/\text{U}$ values, the most convenient solution is selected by weighting the benefits on time saving versus the loss of accuracy, as in the case of Figure 7. In this example, the selected solution is characterised by $d_i = 10$ and
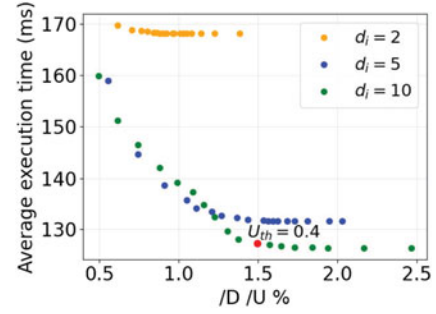

FIGURE 7. An example plot of the various configurations.

$U_{th} = 0.4$, as reported in the figure. If the different combinations of d_i and U_{th} values generate solutions that heavily impact $\%/\text{D}/\text{U}$, the selection of the final solution is driven by a maximum $\%/\text{D}/\text{U}$ value the designer can afford, identifying the corresponding d_i and U_{th} values of the CNN.

E. CNN PRUNING
The pruning activity aims at reducing the size of a CNN by removing redundant connections, neurons and convolutional filters; the effect is a considerably lower execution time of the checker. We apply to the chosen CNN two state-of-the-art strategies to reduce the amount of redundant connections in Neural Networks: *Average Percentage of Zeros* (APoZ) and *Sum of Absolute Weights* (SoAW) [34]. After the execution of the pruning, the obtained CNN has to be first retrained to adjust the weights of the remaining connections and re-evaluated in terms of its classification accuracy. The three steps (pruning, re-training and evaluation) are repeated several times until the difference between the accuracy of the pruned CNN and of the original one is below a given threshold, i.e., as long as the pruning does not invalidate the effectiveness of the checker.

Ideally, the pruning should be applied before the selection of the final configuration in the solution space; this would guarantee the selection of the best solution. However, since the pruning activity is extremely time consuming, reversing the order of the two steps is not affordable time-wise, because of the number of possibilities. Nonetheless, we empirically noticed that in general the proposed order of the operations does not affect the final result.

F. FINAL EVALUATION
The final step consists in the integration of the designed CBs together with the nominal filters to build the final hardened application. At this step, images from the test set are used to evaluate the average execution time and the percentage of misclassified unusable outputs $\%/\text{D}/\text{U}$. This final evaluation is carried out by executing the overall application pipeline a number of times both in fault-free and faulty scenarios. At every iteration, an image from the test set is randomly selected and the pipeline is executed. When the faulty scenario is evaluated, the corrupted intermediate image is "injected" on the output of a stage that is chosen by weighting the probability of each stage to be faulty by its execution

time. The $\%/\text{D}/\text{U}$ is calculated as the ratio between the number of $/\text{D}/\text{U}$ and the total number of experiments carried out in the faulty scenario, while the average execution time is calculated as the average execution time of the final hardened application in the fault-free scenario. Indeed, to optimize the overall design time needed to carry out the methodology, it is possible to postpone the generation of the test set to this final step, once the approximation levels for all filters have been selected.

VI. EXPERIMENTAL EVALUATION

This section presents the results of the experimental evaluation of the proposed approach on two case studies.

A. THE CONSIDERED CASE STUDIES

Case Study #1: Buildings Identification in Aerial Photos

Our first application is meant for the identification of buildings in aerial photos. The application receives a bitmap and produces a heatmap of the same size where each pixel represents the probability of the corresponding pixel in the input image to belong to a building. The application is composed of four steps: i) a *sharpening* filter; ii) a *thresholding* stage that performs a pixel-wise classification based on specific values for each pixel channel; iii) a *reshape&convolution (R&C)* stage that performs a classification based on image reshaping and then a convolution. The two previous classification steps take the sharpened image and produce two probability matrices. Finally, iv) an *aggregation* stage performs a pixel-wise multiplication between the outputs of the two previous steps to generate the output heatmap. As a simple downstream application, we implemented an Oracle that, given a (possibly faulty) heatmap and the corresponding pristine counterpart, outputs a *usable* label if the areas identified as buildings in the two heatmaps overlap for at least the 85%. To build *TRS*, *VS* and *TES*, we employed 1,000 1080x720 images downloaded from MS Bing Maps.

Case Study #2: Motion Detection in Highway Videos

The second case study is an application detecting moving cars in images taken by a camera on the highway. The application produces the list of coordinates of the bounding boxes around the identified cars. The application is composed of four steps: i) an *RGB to gray scale* conversion; ii) a *Gaussian* filter to remove the noise in the image; iii) a *Gaussian Mixture Model (GMM)* stage producing a black/white image reporting the blobs on the moving cars in the scene; and iv) an *Erosion* filter to improve the shape of the blobs. Finally, a *bounding box* detector computes the coordinates of the bounding boxes and draws them around the blobs. This last step has to not been considered in this study; in fact, since it does not produce an image/heatmap in output, the proposed scheme cannot be applied on it. Thus, it has be hardened with a classical DWC.

The Oracle classifies the usability by checking the number and the position of the bounding boxes based on the Jaccard index as in [35]. To build *TRS*, *VS* and *TES*, we employed 1,000 600x300 images downloaded from Italian highway webcams.

B. EXPERIMENTAL SETUP

A Python tool implements our design flow, automating all methodology steps. In particular, all the datasets are obtained by means of error simulation while CNN training, evaluation and pruning exploit the TensorFlow and Keras frameworks [36]. For the datasets generation, error models are manually defined as in [37]; we performed fault injection campaigns (by means of LLFI [27]) for each filter and we extracted the recurrent distortion patterns.

The CNNs featured by the Control Blocks are constituted by a number of convolutional layers interleaved with max-pooling layers, followed by two fully-connected layers, the last of which outputs the final classification. All the models are sequential, meaning that each layer takes as input the output of the previous one and feeds the next one with its output. All the convolutional layers have kernels of size 2x2 or 3x3 and all the max-pooling layers have a 2x2 kernel. Finally, all the activation functions are ReLU with the exception of the last layer, which for classification purposes uses a Sigmoid activation. The downscaling module has been implemented as an average-pooling since it offers good execution time while preserving the relevant information in the image, with the only exceptions of the Gaussian Mixture Model and Erosion filters where given the type of the elaboration the max-pooling is more appropriate. Finally, we chose all possible divisors of the image sizes as d_i values in order not to introduce padding.

Both the nominal and the hardened image processing applications have been implemented in C++. The required execution times have been profiled on an Odroid XU3 Board, pinning the application on a single-core Arm A15 core working at 1.6GHz.

C. EXPERIMENTAL RESULTS

We here report the results of the application of our approach to the two case studies. First, we present the execution time required by the design flow to achieve the final implementation of the checkers. We sequentially executed all the activities of the design flow on a home personal computer equipped with an i7-7700HQ CPU, 16 GByte RAM and an NVIDIA GeForce GTX 1050 (4 GByte dedicated memory). Considering Case Study #1, the design flow took about 180 hours overall; in details: 36 hours for the generation of all the training, validation and test sets, 2 hours for the training of each one of the three considered CNNs for each one of the four application stages and 30 hours for the pruning of the four selected CNNs (one for each application stage). Similar considerations can be drawn for Case Study #2, for which the proposed design flow took about 45 hours (the significantly shorter time is due to the reduced size of the input images w.r.t. Case Study #1).

Then, we present the peculiarities of the solution space definition and exploration, and the comparison of the final selected hardened implementations against the two adopted baselines, i.e., the traditional DWC and the solution in [7]. For each one of them, we present:
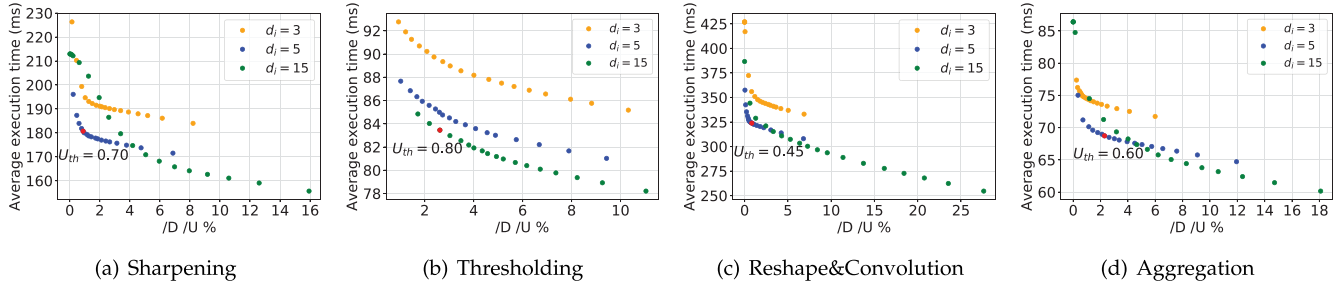
(a) Sharpening   (b) Thresholding   (c) Reshape&Convolution   (d) Aggregation

FIGURE 8. Case study #1: best configuration selection for each filter.

TABLE 2. Stage output classification and handling in the selected configuration for the SCs of case study #1.

Stage	Fault-free execution				Faulty execution			
	D /U	/D U	D U	/D /U	D /U	/D U	D U	/D /U
Sharp.	0.00%	99.80%	0.20%	0.00%	55.89%	41.07%	2.22%	0.82%
R&C	0.00%	98.70%	1.30%	0.00%	54.36%	41.75%	3.10%	0.79%
Thresh.	0.00%	93.90%	6.10%	0.00%	47.66%	41.35%	8.51%	2.48%
Aggreg.	0.00%	99.20%	0.80%	0.00%	39.62%	54.52%	4.92%	0.94%

- For each step of the application, the solution spaces, where each point corresponds to a solution with a U_{th} value and a downscaling factor: Figure 8 for Case Study #1, and Figure 9 for Case Study #2, respectively. The selected best configuration is marked in red reporting also the U_{th} value.
- For each stage, the distribution of discarded/not discarded, usable/unusable cases characterizing the selected configurations after pruning to implement the SC and CB: Tables 2 and 5.
- For each stage, the execution times of the nominal filter, of the corresponding duplicated approximated filters in the CB and of the SC: Tables 3 and 6.
- The comparison of our approach against the DWC and [7] in terms of average execution time and percentage of false negatives when the complete hardened application is considered: Tables 4 and 7.

The analysis of the results allows to draw the following considerations. As it can be noticed, in general, different levels of approximation generate partially overlapping sets of points, and consequently none of the approximation levels dominates the others from the Pareto dominance point of view. Thus, the best configuration has been selected among the subset of Pareto dominant points in order to limit the number of /D /U and at the same time minimize the

execution time. In case the values on the %/D /U axis are limited to a small range close to the origin, as for the *Gaussian* filter in Case Study #2, we selected the point right before the curve becomes flat (i.e., the "elbow" of the curve). In fact, further decreasing the threshold U_{th} of the classification model causes an increase of the %/D /U that is higher than what we deemed tolerable to achieve an only marginal extra time saving. In other cases, as for all filters in Case Study #1, the points are distributed on a large range of %/D /U values; therefore, given our priority to limit the percentage of false negatives, in these experimental sessions we assumed a maximum %/D /U equal to 3% and based on that we identified the best configuration.

Then, considering the classification distribution for the selected configuration presented in Tables 2 and 5, we may notice that the proposed approach presents false positives (D U cases) also in the fault-free scenario (due to approximation and the use of CNNs). This means that also in absence of faults, the SC may trigger unnecessary re-executions. Indeed, the amount of triggered re-executions heavily depends on the complexity of the filter and its susceptibility to the downscaling of the input. However, as it will be shown later, this does not prevent the proposed approach to still provide a significant time saving (again, due to approximation). This is due to the fact that, as it can be observed from Tables 3 and 6, the execution times of the the nominal filters

TABLE 3. Execution times for each stage of case study #1.

Stage	Nominal	Approx.	SC
Sharp.	97.00 ms	11.30 ms	16.31 ms
R&C	184.00 ms	17.38 ms	6.48 ms
Thresh.	32.00 ms	3.80 ms	4.80 ms
Aggreg.	34.00 ms	5.70 ms	6.46 ms

TABLE 4. Comparative analysis of the fully hardened case study #1.

Approach	Avg. Time	/D /U
Our approach	414.40 ms	0.90%
DWC	652.80 ms	0.00%
[7]	652.80 ms	0.20%

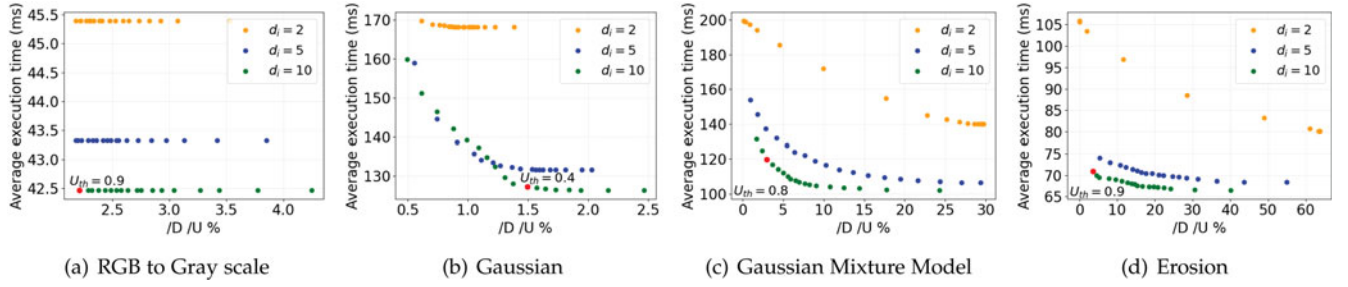(a) RGB to Gray scale   (b) Gaussian   (c) Gaussian Mixture Model   (d) Erosion

FIGURE 9. Case study #2: best configuration selection for each filter.

TABLE 5. Stage output classification and handling in the selected configuration for the SCs of case study #2.

Stage	Fault-free execution				Faulty execution			
	D /U	/D U	D U	/D /U	D /U	/D U	D U	/D /U
RGB-Gray	0.00%	100.00%	0.00%	0.00%	62.85%	33.65%	1.09%	2.41%
Gaussian	0.00%	90.20%	9.80%	0.00%	38.90%	53.69%	5.77%	1.64%
GMM	0.00%	73.26%	26.73%	0.00%	26.20%	51.23%	19.04%	3.53%
Erosion	0.00%	80.00%	20.00%	0.00%	59.69%	28.34%	8.01%	3.96%

is in general considerably larger than the ones of their approximated counterparts hardened with DWC and of the corresponding SC, thus leading to significant time savings although the unnecessary re-executions.

We also performed an in-depth analysis of the intermediate and final results of several runs of each case study, which confirmed the discussion reported in Section III.C. In particular, by referring to the example images in Figure 2 we can say that, as expected, in the case of no fault occurrence (Figure 2(b)), as well as in the case of a fault that slightly corrupts the intermediate output (Figure 2(f)) our approach correctly continues the execution of the application without discarding the output. When a fault causes the disruption of the intermediate output (Figure 2(d)) our approach correctly discards the output and triggers a re-execution.

Finally, we can analyse the execution time and the percentage of false negatives for the complete hardened applications

TABLE 6. Execution times for each stage of case study #2.

Stage	Nominal	Approx.	SC
RGB-Gray	2.44 ms	0.71 ms	1.39 ms
Gaussian	84.90 ms	0.79 ms	1.47 ms
GMM	60.46 ms	1.81 ms	1.38 ms
Erosion	25.70 ms	1.15 ms	4.02 ms

TABLE 7. Comparative analysis of the fully hardened case study #2.

Approach	Avg. Time	/D /U
Our approach	231.68 ms	3.70%
DWC	347.80 ms	0.00%
[7]	347.80 ms	1.30%

and we can compare these results with the DWC and [7]. In particular, when looking at Tables 4 and 7 we can see that we achieve time savings ranging of 36.52% and 33.39% w.r. t. the DWC and [7], for the two case studies.[3] When taking into account the number of false negatives (/D /U), it ranges from 0.90% and 3.70% for the two case studies. This result is an absolute loss w.r.t. the DWC approach that never fails, but is comparable w.r.t. [7], deemed acceptable for not critical tasks.

As a final consideration, we report the memory occupation for the proposed solution. The memory footprint of the final pruned CNNs adopted to harden the considered applications is about 150KByte on average. Therefore, the total occupation of all the checkers is about 600KByte. By considering that the memory occupation for Case Study #1 is about 40MByte, and that for Case Study #2 is about 12MByte, we have an area occupation overhead of about 1.5% and 5%, respectively. This overhead is totally comparable with that of the baseline in [7]; on the other hand, the memory overhead introduced by the TRC employed in the classical DWC is negligible.

VII. CONCLUSION

We presented a novel lightweight fault tolerant scheme specifically tailored for image processing applications. The proposed scheme enhances the classical DWC by: i) replacing one of the two exact application replicas with an approximated counterpart, and ii) adopting a usability-based classification of (intermediate) outputs through the use of Convolutional Neural Network-based checkers. A companion design methodology supports the designer in the application and tuning of the

---

[3]Note that in the fault free scenario the behaviour (and therefore the execution time) of the DWC and [7] is the same.

proposed scheme to optimize both the performance and the fault tolerance capabilities of the hardened system.

The approach has been applied to two case studies; it achieves time savings larger than 30% w.r.t. the traditional DWC with re-execution approach and the proposal in [7]. At the same time, the percentage of false negatives is always lower than 4%, acceptably low and comparable with the one in [7]. Future work will investigate the application of the proposed approach on different HW platforms, the capability to deal with permanent faults and will better refine the approximation-based hardening strategy, for instance in the direction of replacing both the replica and the checker with a CNN.

REFERENCES

[1] J. Andersson, M. Hjorth, F. Johansson, and S. Habinc, "LEON processor devices for space missions: First 20 years of LEON in space," in *Proc. Int. Conf. Space Mission Challenges Inform. Tech.*, 2017, pp. 136–141.

[2] M. Fayyaz and T. Vladimirova, "Fault-tolerant distributed approach to satellite on-board computer design," in *Proc. Aerosp. Conf.*, 2014, pp. 1–12.

[3] L. Sterpone, M. Porrmann, and J. Hagemeyer, "A novel fault tolerant and runtime reconfigurable platform for satellite payload processing," *IEEE Trans. Comput.*, vol. 62, no. 8, pp. 1508–1525, Aug. 2013.

[4] R. L. Davidson and C. P. Bridges, "Error resilient GPU accelerated image processing for space applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 9, pp. 1990–2003, Sep. 2018.

[5] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, 2016.

[6] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proc. Des. Autom. Conf.*, 2015, pp. 1–6.

[7] M. Biasielli, C. Bolchini, L. Cassano, E. Koyuncu, and A. Miele, "A neural network based fault management scheme for reliable image processing," *IEEE Trans. Comput.*, vol. 69, no. 5, pp. 764–776, May 2020.

[8] I. Albandes, et al., "Improving approximate-TMR using multi-objective optimization genetic algorithm," in *Proc. Latin-Amer. Test Symp.*, 2018, pp. 1–6.

[9] A. J. Sanchez-Clemente, L. Entrena, R. Hrbacek, and L. Sekanina, "Error mitigation using approximate logic circuits: A comparison of probabilistic and evolutionary approaches," *IEEE Trans. Rel.*, vol. 65, no. 4, pp. 1871–1883, Dec. 2016.

[10] A. Ullah, P. Reviriego, S. Pontarelli, and J. A. Maestro, "Majority voting-based reduced precision redundancy adders," *IEEE Trans. Device Mater. Rel.*, vol. 18, no. 1, pp. 122–124, Mar. 2018.

[11] R. Hegde and N. R. Shanbhag, "Soft digital signal processing," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 9, no. 6, pp. 813–823, Dec. 2001.

[12] B. Shim and N. R. Shanbhag, "Energy-efficient soft error-tolerant digital signal processing," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 14, no. 4, pp. 336–348, Apr. 2006.

[13] A. Raha and V. Raghunathan, "Towards full-system energy-accuracy tradeoffs: A case study of an approximate smart camera system," in *Proc. Des. Autom. Conf.*, 2017, pp. 1–6.

[14] M. Biasielli, L. Cassano, and A. Miele, "An approximation-based fault detection scheme for image processing applications," in *Proc. Des. Autom. Test Europe Conf.*, 2020, pp. 1331–1334.

[15] I. A. C. Gomes, M. G. A. Martins, A. I. Reis, and F. LimaKastensmidt, "Exploring the use of approximate TMR to mask transient faults in logic with low area overhead," *Microelectron. Rel.*, vol. 55, no. 9, pp. 2072–2076, 2015.

[16] M. R. Choudhury and K. Mohanram, "Approximate logic circuits for low overhead, non-intrusive concurrent error detection," in *Proc. Des. Autom. Test Eur. Conf.*, 2008, pp. 903–908.

[17] M. R. Choudhury and K. Mohanram, "Low cost concurrent error masking using approximate logic circuits," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 32, no. 8, pp. 1163–1176, Aug. 2013.

[18] K. Chen, J. Han, and F. Lombardi, "Two approximate voting schemes for reliable computing," *IEEE Trans. Comput.*, vol. 66, no. 7, pp. 1227–1239, Jul. 2017.

[19] B. Pratt, M. Fuller, and M. Wirthlin, "Reduced-precision redundancy on FPGAs," *Int. J. Reconfigurable Comput.*, vol. 2011, pp. 1–12, 2011.

[20] G. S. Rodrigues, J. Fonseca, F. Benevenuti, F. LimaKastensmidt, and A. Bosio, "Exploiting approximate computing for low-cost fault tolerant architectures," in *Proc. Symp. Integr. Circuits Syst. Des.*, 2019, pp. 1–6.

[21] B. Shim, S. R. Sridhara, and N. R. Shanbhag, "Reliable low-power digital signal processing via reduced precision redundancy," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 12, no. 5, pp. 497–510, May 2004.

[22] F. Baharvand and S. G. Miremadi, "LEXACT: Low energy N-Modular redundancy using approximate computing for real-time multicore processors," *IEEE Trans. Emerging Top. Comput.*, vol. 8, no. 2, pp. 431–441, Apr.–Jun. 2020.

[23] G. S. Rodrigues, A. Barrosde Oliveira, A. Bosio, F. L. Kastensmidt, and E. Pignatonde Freitas, "ARFT: An approximative redundant technique for fault tolerance," in *Proc. Conf. Des. Circuits Integr. Syst.*, 2018, pp. 1–6.

[24] C. Bolchini, L. Cassano, A. Miele, and M. Biasielli, "Lightweight fault detection and management for image restoration," in *Proc. Int. Symp. Defect Fault Tolerance Very Large Scale Integr. Syst.*, 2020, pp. 1–6.

[25] G. *et al.*, "Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 35, no. 12, pp. 44–56, Dec. 2020.

[26] A. Biondi, F. Nesti, G. Cicero, D. Casini, and G. Buttazzo, "A safe, secure, and predictable software architecture for deep learning in safety-critical systems," *IEEE Embedded Syst. Lett.*, vol. 12, no. 3, pp. 78–82, Sep. 2020.

[27] J. Wei, A. Thomas, G. Li, and K. Pattabiraman, "Quantifying the accuracy of high-level fault injection techniques for hardware faults," in *Proc. Intl. Conf. Dep. Syst. Netw.*, 2014, pp. 375–382.

[28] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, Poster, 2014.

[29] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.

[30] G. Hinton, "Overview of mini-batch gradient descent." Accessed: Dec. 14, 2020. [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[31] R. Caruana, S. Lawrence, and C. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Proc. Neural Inf. Process. Syst.*, 2001, pp. 402–408.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.

[33] S. M. Kakade, S. Shalev-Shwartz, and A. Tewari, "Regularization techniques for learning with matrices," *J. Mach. Learn. Res.*, vol. 13, pp. 1865–1890, 2012.

[34] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," in *Proc. IEEE Intl. Conf. Comput. Vis.*, 2017, pp. 5068–5076.

[35] F. Fernandes dosSantos, L. Carro, and P. Rech, "Kernel and layer vulnerability factor to evaluate object detection reliability in GPUs," *IET Comput. Digi. Techn.*, vol. 13, no. 3, pp. 178–186, 2019.

[36] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems." Accessed: Jul. 7, 2021. [Online]. Available: www.tensorflow.org/

[37] C. Bolchini, L. Cassano, A. Mazzeo, and A. Miele, "Error modeling for image processing filters accelerated onto SRAM-based FPGAs," in *Proc. Intl. Symp. On-Line Testing Robust Syst. Des.*, 2020, pp. 1–6.

MATTEO BIASIELLI received the MSc degree in computer science engineering from the University of Illinois at Chicago and from Politecnico di Milano in 2019. His research interests include artificial intelligence and machine learning, working on solutions for fault tolerance in image processing applications.

CRISTIANA BOLCHINI (Senior Member, IEEE) received the PhD degree in automation and computer engineering from Politecnico di Milano in 1997. She is currently a professor with Politecnico di Milano. She has coauthored more than 130 papers in peer-reviewed international journals and conferences. Her research focuses on methodologies for the design and analysis of digital systems with a specific focus on dependability and self-awareness for heterogeneous system architectures. In 2019, she was the technical programme chair of the conference IEEE/ACM Design Automation and Test in Europe.
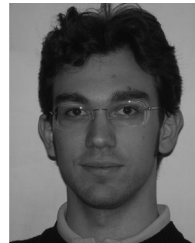
ANDREA MAZZEO received the MSc degree in 2019 in computer science engineering from Politecnico di Milano, where he is currently working toward the PhD degree in computer science engineering. His research interests include machine learning, deep learning, re-configurable hardware devices, and the analysis and the design of fault-tolerant image processing applications.

LUCA CASSANO (Member, IEEE) received the BS, MSc and PhD degrees in computer engineering from the University of Pisa, Italy. He is currently an assistant professor with Politecnico di Milano, Italy. His research interests include definition of innovative techniques for fault simulation, testing, untestability analysis, diagnosis, and verification of fault tolerant and secure digital circuits and systems. With the PhD thesis, titled "Analysis and Test of the Effects of Single Event Upsets Affecting the Configuration Memory of SRAM-based FPGAs", he was the recipient of the European semifinals of the 2014 TTTC's E. J. McCluskey Doctoral Thesis Award and he classified as runner-up at the world finals.

ANTONIO MIELE (Senior Member, IEEE) received the MSc degree in computer engineering from Politecnico di Milano, a the MSc in computer science from the University of Illinois at Chicago, and the PhD degree in information technology from Politecnico di Milano. He is currently an associate professor with Politecnico di Milano. He has coauthored more than 80 peer-reviewed publications. His main research interests include design methodologies for embedded systems, in particular fault tolerance and reliability issues, runtime resource management in heterogeneous multi-/many-core systems, and FPGA-based systems design.