Lightweight Fault Detection and Management for Image Restoration

Cristiana Bolchini[1], Luca Cassano[1], Antonio Miele[1], Matteo Biasielli[2*]
[1]*Dipartimento di Elettronica, Informazione e Bioingegneria – Politecnico di Milano*, Italy
[2]*King Digital Entertainment plc.* – Stockholm, Sweden
[1]{first_name.last_name}@polimi.it, [2]matteo.biasielli@king.com

***Abstract*—Image restoration is generally employed to recover an image that has been blurred, for example, for noise suppression purposes. The Richardson-Lucy (RL) algorithm is a widely used iterative approach for image restoration. In this paper we propose a lightweight application-specific fault detection and management scheme for RL that exploits two specific characteristics of such algorithm: i) there is a strong correlation between the input and output images of each iteration, and ii) the algorithm is often able to produce a final output that is very similar to the expected one although the output of an intermediate iteration has been corrupted by a fault. The proposed scheme exploits these characteristics to detect the occurrence of a fault without requiring duplication and to determine whether the error in the output of an intermediate iteration of the algorithm would be absorbed (thus avoiding image dropping and algorithm re-execution) or whether the image has to be discarded and the overall elaboration to be re-executed. An experimental campaign demonstrated that our scheme allows for an execution time reduction of about 54% w.r.t. the classical Duplication with Comparison (DWC), still providing about 99% fault detection.**

***Index Terms*—Fault detection, Fault recovery, Image processing, Image restoration, Richardson-Lucy deconvolution**

I. Introduction and related work

Image processing applications are employed in a variety of scenarios for supporting critical decision processes. Examples are automotive [1], avionics [2], and aerospace [3]. In these scenarios, the computing systems are safety-/mission-critical, therefore it is vital (often required by strict design standards) that the system continues providing its correct functionality also after the occurrence of a fault [4].

At the same time, image processing applications may expose an inherent degree of resiliency due to its intrinsic inexact nature [5]. First, the application might have been designed to process inputs that are noisy (e.g., images coming from sensors). Further, the output of the application may be a probabilistic estimate, e.g., in machine learning-based applications. Finally, the end-user of the application's output may effectively carry out its task although the input image is slightly altered.

When reliability is a need, image processing applications have to be hardened with specific fault detection and management mechanisms. However, classical schemes, such as Duplication with Comparison (DWC) or Triple Modular Redundancy (TMR), do not suffice because the considerable overhead they introduce conflicts with the data- and compute-intensive nature and strict performance requirements of such applications. Moreover, a bit-wise checking, as the one performed by the DWC, may be too stringent when considering the intrinsic inexactness of such applications. Indeed, discarding the output as soon as the redundant output images differ for a single pixel represents an unnecessary performance degradation, since *slightly* altered images might be effectively exploited by the end-user application [6].

Several strategies have been proposed to exploit such peculiarities of image processing applications to reduce the hardening cost. In [7] DWC has been selectively applied to the critical portions of considered pedestrian detection application. In [8], based on the idea of inexact computation, approximate computing techniques have been applied to TMR to reduce the overhead. Nevertheless, this approach works at the granularity of the single bit (the entire image is only considered in the validation phase). A similar idea for fault detection purposes in the signal processing domain has been proposed in [9] to check single scalar values. In this approach, based on the characteristics of the application under analysis, the redundant replica is replaced with a smaller estimator or even removed, i.e., the application's input is used to feed the checker in place of the redundant outputs. Recently, a more advanced approach has been proposed in [6] where a corrupted image is globally checked, and possibly discarded, based on its usability w.r.t. the end-user application. This approach has been later enhanced in [10] by approximating the replica, thus achieving a considerable performance improvement.

In this paper we aim at exploring the definition of application-specific fault detection and management for the image processing scenario, considering the specific scenario of the Richardson-Lucy (RL) algorithm [11], [12], a widely-used and compute-intensive iterative algorithm for restoration of blurred images [13], [14]. We propose an approach that, by only analyzing the input and output images of each RL iteration, determines whether a fault occurred and whether it corrupted the image in such a way that the final output will be not usable, eventually triggering a re-computation.

Our proposal exploits two specific characteristics of the RL algorithm that have been highlighted through an in-depth fault injection campaign: i) there is a strong correlation between the input and output images of each algorithm iteration, and ii) the effect of faults during the processing is often absorbed

*Matteo Biasielli's current affiliation is *King Digital Entertainment plc.*, but the entirety of his contribution to the present work has been carried out within the context of his previous affiliation, *Politecnico di Milano.*

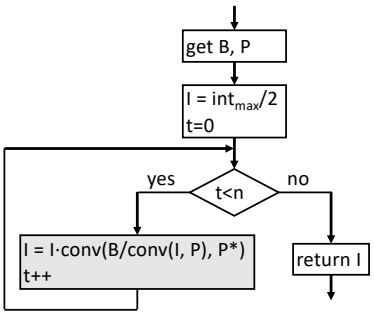


Figure 1. Overall scheme of Richardson-Lucy (RL) algorithm.

and thus the algorithm is able to produce a final output that is very similar to the expected one. Thanks to such peculiarities, the proposed fault detection and management scheme is able to i) detect faults without requiring any replication, and ii) trigger a re-execution of the application only if the corrupted intermediate output will cause the final output to be unusable. To summarize, the main contributions of this paper are:

- an in-depth analysis of the behavior of the RL algorithm in the presence of faults (Section III);
- a lightweight fault detection and management scheme exploiting a Convolutional Neural Network (CNN) classifier specifically tailored for the RL algorithm (Section IV);
- an experimental evaluation of the approach reporting an average execution time reduction of about 54% w.r.t. the classical DWC, still providing about 99% fault detection.

The remainder of this paper is organized as follows. The next section presents the RL algorithm and the working scenario. The RL algorithm is systematically analyzed in Section III to characterize its response to faults; the fault detection and management scheme is discussed in the subsequent section. Section V presents an experimental evaluation of the proposed scheme, while Section VI draws the conclusions.

II. Background

A. The Richardson-Lucy (RL) algorithm

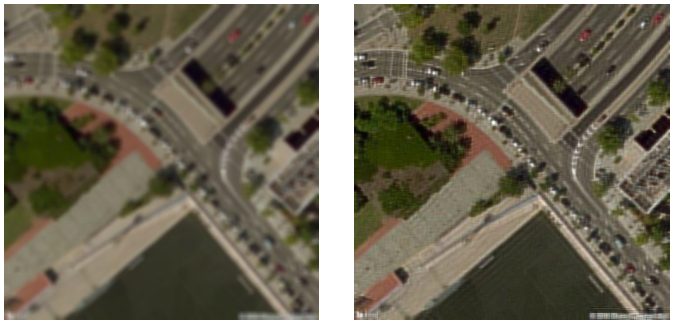
The Richardson-Lucy (RL) algorithm [11], [12] is a deconvolution method originally designed in the field of astronomical image restoration and nowadays exploited more broadly for recovering blurred images, for instance, for noise suppression. The method takes in input a blurred image B, being a bi-dimensional matrix of pixel intensity values[1], and assumes an a-priori known point spread function P, i.e. the bi-dimensional matrix causing the blurring effect. The algorithm produces the estimated clear image I by applying iteratively Bayesian estimation according to the following formula:

$$I^{t+1} = I^t \cdot \left(\frac{B}{I^t \circledast P} \circledast P^* \right) \quad (1)$$

where $\cdot$ and $/$ represent scalar multiplication/division and $\circledast$ a 2D convolution, and P^* is the flipped version of P.

Figure 1 depicts the RL algorithm; it takes in input a blurred image (in Figure 2(a)) and starts with an estimated clear image

[1]For colored images, the algorithm is applied on each channel separately.



(a) Input (b) Output

Figure 2. RL input and output images in a fault-free execution.

I having all pixels assigned to a given value, for instance a middle-scale intensity value, and applies iteratively Equation 1 for a given number n of *iterations*. At the end of the iterations, the final I is the final restored image (in Figure 2(b)).

It can be noticed from Equation 1 that the single iteration of the algorithm is highly computational-intensive, since it comprises two convolutions and two scalar operations. Moreover, depending on n, the equations may be applied for a large number of times, thus generating a long processing pipeline. As a consequence, for this kind of applications the quest for more efficient hardening schemes is particularly necessary.

B. Working Scenario

In this work we consider a software implementation of the RL algorithm running on a general purpose embedded CPU. In this scenario, the CPU may be affected by Single Event Upsets (SEUs) that may cause the output of the elaboration to be corrupted. SEUs in CPUs may lead to i) application crashes or operating system exceptions blocking the execution, ii) application hangs leading to a non-termination, and iii) silent data corruption, where the executed application terminates producing an erroneous output, without any alert [15]. The first two classes of effects can be effectively and autonomously managed by the operating system through watchdog mechanisms. On the other hand, silent data corruption requires ad-hoc hardening techniques, addressed here. Finally, we assume that a single fault may occur during a single application run.

C. Baseline hardening scheme

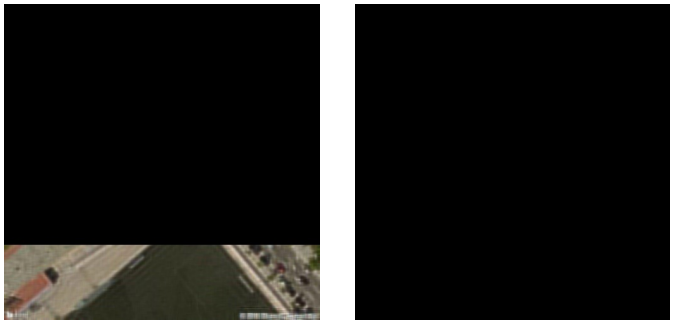
The baseline reference for fault detection and management in a CPU-based system is the adoption of time redundancy, consisting in a DWC triggering re-execution of the application when the fault causes an error. We consider DWC applied at the granularity of the single iteration of the RL algorithm, checking redundant iteration outputs by means of a Two-Rail Checker (TRC). On the detection of a single pixel mismatch the pipeline is re-executed.

D. Usability-based checking

We adopt the usability-based fault detection paradigm, proposed in [6], to overcome the discussed limitation of DWC when applied in the image processing scenario. Results of an elaborations (either the intermediate or the final ones) are

(a) Output at iteration 72 (b) Final output

Figure 3. RL intermediate and output images in a faulty execution where the output is usable.



(a) Output at iteration 67 (b) Final output

Figure 4. RL intermediate and output images in a faulty execution where the output is unusable.

globally checked by considering the usability of the produced output from the end-user application point of view.

As an example of end-user application for the RL algorithm, we considered in this paper the application for building identification presented in [6]. The application takes deblurred aerial image produced by RL algorithm and, based on four kernels, it identifies the buildings in the image. To determine whether a corrupted RL output is usable by such end-user application, we run the building identification application on the corrupted image and on the corresponding golden one: if the identified buildings overlap for at least 95%, the corrupted input image is assumed to be usable, otherwise unusable.

Figure 3 reports the intermediate and final output images in a case where the fault caused a slight alteration in the output of the computation (a small black square in the middle of the image). In this case the final output of RL was usable by the building identification application. Conversely, Figure 4 shows the case where the fault disrupted the output of the computation, making it unusable for the building identification.

III. Reliability characterization of the Richardson-Lucy algorithm

We analyzed a software implementation of the RL algorithm through fault injection (by means of LLFI [15]) to characterize its behavior in the presence of faults during the computation. The outcome of this experiment is twofold: on the one hand we analysed the difference between the input and output of each iteration to quantitatively measure their correlation both in fault-free and in faulty scenarios. On the other hand, we observed how the RL algorithm reacts to the occurrence of faults and how faults affect the final output of the algorithm.

A. Correlation between iteration's input and output images

Since every iteration of the RL algorithm produces slight modifications in the iteration's output w.r.t. the iteration's input, we measured the distance between these two images at each iteration in terms of Root Mean Square Error (RMSE).

First we performed 5,000 fault-free RL runs each with a different input image and we measured the RMSE between the input and the output of each iteration t for each application run. This analysis reported that the distance between iteration's input and output images is always a monotone decreasing function. Figure 5(a) reports an example of such analysis for a fault-free run of RL with $n = 100$ on the image in Figure 2(a).

Then, we analysed how the regularity of this measure is perturbated by the occurrence of faults. We performed a large random fault injection campaign so to collect 10,000 experiments where the produced iteration's output had been corrupted. Again we analyzed the RMSE between the iteration's input and output images at each iteration t for each experiment. This analysis demonstrated that the RMSE curve presents a spike corresponding to the iteration in which the fault has been injected. Of course, the height of the spike depends on the magnitude of the effect of the fault in the iteration's output image. Figure 5(b) reports the case of a fault causing a small black square in the iteration's output (see Figure 3) where the RMSE increases of about 1.5, while Figure 5(c) reports the case of a fault causing a large black area (see Figure 4) where the RMSE increases of about 80.

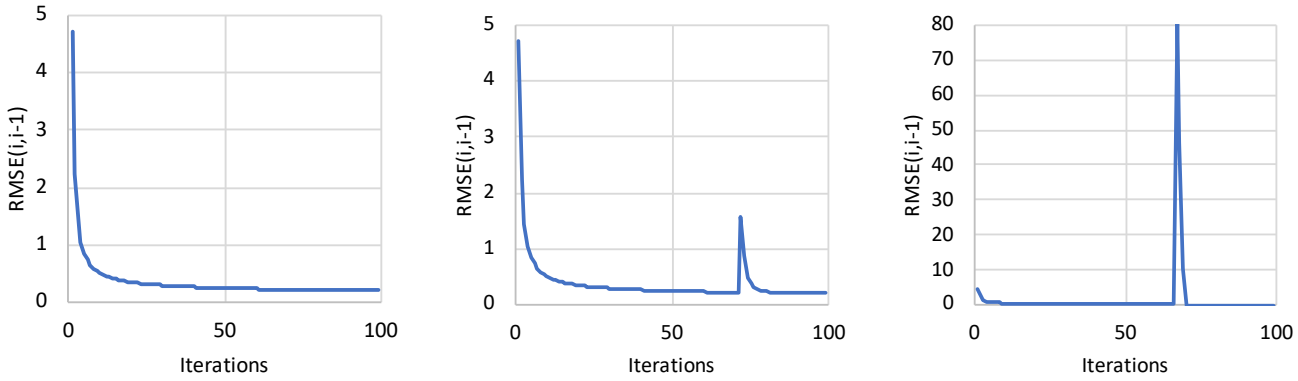
B. Propagation of errors through multiple iterations

We also investigated the propagation of the effect of faults through the subsequent iterations of the RL algorithm and we measured the impact on the final output in terms its usability by the considered building identification application.

A first scenario is when the occurred fault has a limited impact on the intermediate image and it remains unaltered until the last iteration. As a consequence, the impact of the fault on the final output is also limited. This is the case depicted in Figure 3 where the output at iteration 72 appears damaged by a small black square in the center of the image. The black square appears more or less unaltered also in the final output, which is therefore usable.

A second scenario is when the occurred fault has a huge impact on the intermediate output and it is amplified by the subsequent iterations. This is the case of Figure 4 where the output at iteration 67 appears heavily damaged. In this case the black rectangle is expanded by the computations of the subsequent iterations and the final output is totally unusable.

A last scenario is when the occurred fault has a significant impact on the intermediate output but it is absorbed by the subsequent iterations, thus allowing the application to produce a usable output. This is the case of Figure 6 where the output at



(a) Golden output (b) Fault effect: black spot (c) Fault effect: black area

Figure 5. Three examples RMSE between iteration's input and output images in fault-free and faulty scenarios.

(a) Output at iteration 8 (b) Final output

Figure 6. RL intermediate and output images in a faulty execution where the output is usable.

(a) Output at iteration 90 (b) Final output

Figure 7. RL intermediate and output images in a faulty execution where the output is unusable.

iteration 8 appears heavily damaged by a shift. Nevertheless, the subsequent iterations largely mitigate this effect and the final output is to be usable. It has to be mentioned that such error masking requires a sufficient number of iterations to be effective, otherwise the effect of the fault will still be visible in the final output. This is the case of Figure 7 where the output at iteration 90 appears heavily damaged by a shift and the final output appears only partially restored. This output is indeed unusable by the building identification application.

Finally, we performed an accurate visual inspection of the outputs of the iterations subsequent the occurrence of the fault. We noticed that it is already possible to determine whether the RL algorithm will absorb the error or not by analysing the outputs of few iterations after the fault occurrence.

IV. The Proposed Hardening Scheme

The proposed approach is an advanced fault detection and management scheme that exploits the previously presented peculiarities of the RL algorithm. The scheme is intended to be applied at the granularity of the single iteration of the algorithm, and aims at overcoming two limitations of the classical DWC scheme: i) the considerable performance overhead due to duplication in a classical DWC applied to software; and ii) the inefficiency due to the strict pixel-wise checking performed by the TRC. In particular, the proposed scheme builds upon

- the decreasing distance between input and output images in two subsequent iterations; and
- the ability of the RL algorithm to produce a usable output although the output of an intermediate iteration suffered from the occurrence of a fault.

The proposed scheme exploits these characteristics to avoid replication and to reduce unnecessary re-executions. Indeed, the iteration's input image (instead of a redundant replica's output) is checked with the iteration's output image. Moreover, after the identification of the occurrence of a fault, a CNN-based classifier is exploited to predict whether the final output will be usable or not.

A high-level representation of the proposed solution is depicted in Figure 8. It consists of two modules, dubbed the *fault detection* and the *fault management* module, respectively. The former computes the RMSE between the input and output images at the current iteration t, dubbed $RMSE(t)$, and compares the obtained value with the one computed at the previous iteration, $RMSE(t-1)$. If $RMSE(t)$ is smaller than $RMSE(t-1)$, the fault detection module assumes that no fault occurred and the processing of the RL algorithm continues. If an increase is observed in $RMSE(t)$ w.r.t. the previous value,

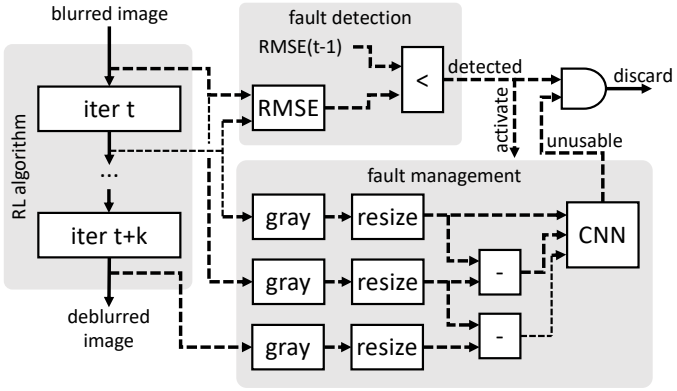


Figure 8. The proposed fault detection scheme.

a fault is assumed to have occurred and the fault management module is executed.

The fault management module features a CNN specifically trained to evaluate the magnitude of the error and to infer whether the error would cause the final application's output to be unusable or not. When the CNN classifies the image as unusable, the overall application is re-executed. Although the entire application is re-executed when the intermediate output is classified as unusable, the checking is performed after any iteration (instead of at the end of the entire application) to halt the faulty execution as soon as possible.

A number of preliminary computations are performed to prepare the CNN inputs. First, k RL iterations after the detection of the fault are executed. Then, images at iterations t, $t-1$ and $t+k$ are converted in gray scale and resized to a smaller dimension. Grey-scaling and resizing are performed to reduce the CNN complexity and its execution time still providing high classification accuracy. The input matrix fed to the CNN is obtained by stacking the grey-scale versions of:

- the image at iteration t,
- the pixel-wise subtraction of images at t and $t-1$, and
- the pixel-wise subtraction of images at $t+k$ and $t-1$.

The image at iteration t and the first pixel-by-pixel subtraction are used to highlight the effect of the fault and its magnitude. Conversely, the second pixel-by-pixel subtraction allows to determine whether the fault effect is going to be absorbed by the remainder of the RL execution.

The advantage of the proposed scheme is twofold: i) the fast fault detection based on the RMSE measurement drastically reduces the execution time w.r.t. the classical DWC, and ii) the use of a CNN to decide whether to discard or not an intermediate image based the usability of the final output (instead of being based on the mere occurrence of a fault as for the TRC) allows to avoid unnecessary re-executions, thus bringing an additional time saving.

Do note that the fault detection module does not detect faults occurring in either the first iteration of the algorithm or the last k one. These iterations have to be protected with a classical time-based duplication. Moreover, although being effective in detecting faults occurring in the processing of the algorithm (the gray box in Figure 1), the proposed scheme is not able to detect faults corrupting the portions of memory storing the following data: i) the iteration counter t, ii) the input image B, and iii) the point spread function P used as filters in the two convolutions. Such unprotected data should be either duplicated or protected by means of error detection/correction codes. Finally, the fault management module is left unprotected because it would be executed only in case a fault has been detected. In the considered single fault scenario this would imply that, when executed, the fault management scheme is not affected by faults and that when affected by a fault it would not be executed. All other faults are covered.

V. Experimental evaluation

We performed an in-depth experimental analysis of the proposed approach to assess its fault detection capability and to measure the introduced benefit in terms of time saving.

A. Experimental setup

We implemented a C++ version of the RL algorithm with the OpenCV library. We considered a Gaussian blur spread point function, with kernel size equal to 10 and different μ and σ values over the two spatial directions (0, 9 and 5, 5, respectively). Finally we tuned 100 algorithm iterations to be enough to restore the original image.

We use a dataset of 1,500 500x500 aerial images downloaded from Bing. Each original image is considered for a specific fault injection campaign with the LLFI fault injection tool [15] to obtain 10 instances where the intermediate output is actually corrupted. We partition the obtained 15,000 corrupted images to generate a final training and a testing sets counting $10,000$ and $5,000$ instances, respectively. As previously discussed, the usability of the corrupted output images has been determined by feeding them into the building identification application [6]. We compared the result of this run with the result produce by the building identification when fed with a golden input. If the amount of overlapping building is above 95% the image is labelled as usable.

We tuned the resize module to produce 100x100 gray-scale images and the k parameter in the fault management module equal to 3. The proposed scheme has been implemented in C++ and integrated in the application. The CNN has been trained with an adapted version of the framework proposed in [6] and automatically translated in C++. We consider a sequential CNN structure consisting of 2+4+2+3 3x3 convolutional layers (respectively with 8, 8, 20, 20, 25, 25 ,25, 30, 30, 25, 20 filters) separated by 2x2 max-pooling layers. The CNN has been automatically optimized by a pruning phase to remove all the unnecessary computations, obtaining only 1,500 parameters from the 49,000 initial ones.

Execution times of the approach and the two baselines (time-redundant DWC and the scheme in [6]) are measured on an ARM A15 core at 1.6GHz hosted on a Samsung Exynos device.

Table I
PERFORMANCE EVALUATION: NOTATION AND CLASSIFICATION

D /U	Correctly handled corrupted image
/D U	Achieved execution time savings
D U	Not exploited execution time savings – false positive
/D /U	Erroneously accepted image – false negative

Table II
FAULT MANAGEMENT CLASSIFICATION ACCURACY AND OVERALL EXECUTION TIMES

	Classification accuracy				Execution times	
	D /U	/D U	D U	/D /U	Fault free	Faulty
DWC	26.08%	0.00%	73.92%	0.00%	73.00s	109.51s
[6]	22.60%	49.96%	23.82%	1.96%	73.00s	88.79s
Our	25.00%	45.12%	28.79%	1.08%	37.97s	50.51s

B. Experimental results

We measured the capability of the proposed approach in i) not raising false alarms; ii) detecting faults and iii) discarding only those corrupted intermediate images that would make the final output unusable. Moreover, we measured the average execution times of the considered RL algorithm when hardened through our proposal.

Fault Detection and Management Analysis: First of all, we analysed the behavior of our proposal in a fault-free scenario. To do so, we fed the $1,500$ original (fault-free) images in the hardened RL algorithm without injecting faults. No false alarm was raised by the fault detection module.

Then, we analysed the ability of the fault detection module in detecting the occurrence of faults. We fed the corrupted images of the test set in the hardened RL algorithm. The fault detection module correctly detects 100% of the occurred faults.

Finally, we assessed the ability of the fault management module in determining whether a previously identified intermediate corrupted image will cause the final output to be unusable. Such measurements have been carried out by feeding the fault management module with the corrupted images of the test set and by counting those classified as too corrupted and therefore to be discarded (D), or those not (/D). This classification is analyzed against what the images really are, usable (U) or unusable (/U) leading to the four classes presented in Table I. Columns 2 to 5 of Table II report the results from this analysis, where the proposed approach is compared with the classical DWC and with the scheme in [6]. It is possible to observe a significant shift from D U to /D U, meaning that the fault management module is actually able to identify the faulty images that will not lead to unusable final outputs. This demonstrates that the proposed approach actually prevents unnecessary re-executions. On the other hand, the amount of erroneously accepted images (/D /U) is only about 1%. When compared against [6], our approach achieves comparable results without introducing any replication.

Time Saving Analysis: We measured the average execution times in the fault-free and in the faulty scenarios for the proposed approach and for the classical DWC and the scheme in [6]. Results from this experiment are reported in the last two columns of Table II. By avoiding replication and by reducing the amount of unnecessary re-executions our approach achieves a 48% and 54% time saving w.r.t. DWC and a 48% and 43% w.r.t. [6] in the fault-free and faulty scenarios, respectively (note that DWC and [6] are equivalent in the fault-free scenario).

VI. CONCLUSIONS

We presented a fault detection and management scheme specifically tailored for the Richardson-Lucy image restoration algorithm. Our scheme significantly reduces reliability-related overhead by i) avoiding replication and ii) exploiting a CNN-based classifier to reduce the number of unnecessary re-executions. An experimental campaign demonstrated that our scheme brings a time saving of about 54% w.r.t. the classical DWC, still providing about 99% fault detection capability.

VII. ACKNOWLEDGMENT

This paper has been partially supported by Intel Corporation under the award titled "Adaptive Application-oriented Fault Detection for Reliable Image Processing". The authors personally thank Professor Giacomo Boracchi, from Politecnico di Milano, for providing the initial idea of applying application-specific fault detection to the Richardson-Lucy algorithm.

REFERENCES

[1] G. Aggarwal and S. Jain, "Road crack detection and segmentation for autonomous driving," in *2019 International Conference on Communication and Electronics Systems (ICCES)*. IEEE, 2019, pp. 198–202.

[2] V. Sudevan, A. Shukla, and H. Karki, "Vision based autonomous landing of an unmanned aerial vehicle on a stationary target," in *2017 17th International Conference on Control, Automation and Systems (ICCAS)*. IEEE, 2017, pp. 362–367.

[3] M. A. Hoffmeyer, K. Miller, I. Balter, and T. J. Roh, "Satellite communications data processing," 2018, US Patent 9,954,602.

[4] International Organization for Standardization, "ISO 26262: Road vehicles – Functional safety," 2011.

[5] S. Mittal, "A Survey of Techniques for Approximate Computing," *ACM Computing Surv.*, vol. 48, no. 4, pp. 62:1–62:33, 2016.

[6] M. Biasielli, C. Bolchini, L. Cassano, E. Koyuncu, and A. Miele, "A Neural Network Based Fault Management Scheme for Reliable Image Processing," *IEEE Trans. on Comp.*, vol. 69, no. 5, pp. 764–776, 2020.

[7] F. Fernandes, L. Weigel, C. Jung, P. Navaux, L. Carro, and P. Rech, "Evaluation of Histogram of Oriented Gradients Soft Errors Criticality for Automotive Applications," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 4, pp. 38:1–38:25, Nov. 2016.

[8] K. Chen, J. Han, and F. Lombardi, "Two Approximate Voting Schemes for Reliable Computing," *IEEE Trans. Computers*, vol. 66, no. 7, pp. 1227–1239, July 2017.

[9] B. Shim and N. R. Shanbhag, "Energy-efficient soft error-tolerant digital signal processing," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 4, pp. 336–348, 2006.

[10] M. Biasielli, L. Cassano, and A. Miele, "An Approximation-based Fault Detection Scheme for Image Processing Applications," in *Proc. Design, Automation Test in Europe Conf.*, 2020, pp. 1331–1334.

[11] W. H. Richardson, "Bayesian-based iterative method of image restoration," *J. Opt. Soc. Am.*, vol. 62, no. 1, pp. 55–59, Jan 1972.

[12] L. B. Lucy, "An iterative technique for the rectification of observed distributions," *Astronomical Journal*, vol. 79, p. 745, Jun. 1974.

[13] N. Kumar, H. Shukla, and R. Tripathi, "Image restoration in noisy free images using fuzzy based median filtering and adaptive particle swarm optimization-richardson-lucy algorithm," *International Journal of Intelligent Engineering and Systems*, vol. 10, no. 4, pp. 50–59, 2017.

[14] J. Jiang, J. Huang, and G. Zhang, "An accelerated motion blurred star restoration based on single image," *IEEE Sensors Journal*, vol. 17, no. 5, pp. 1306–1315, 2016.

[15] 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. Dependable Systems and Networks*, 2014, pp. 375–382.