2020 International Conference on Computing and Information Technology, University of Tabuk, Kingdom of Saudi Arabia.
Volume: 01, Issue: ICCIT- 1441, Page No.: 368 – 373, 9th & 10th Sep. 2020.

Analyzing the Reliability of Convolutional Neural Networks on GPUs: GoogLeNet as a Case Study

Younis Ibrahim
College of IoT Engineering
Hohai University
Changzhou, China
Younis@hhu.edu.cn

Haibin Wang
College of IoT Engineering
Hohai University
Changzhou, China
wanghaibin@hhuc.edu.cn

Khalid Adam
Faculty of Electrical & Electronic Eng.
University Malaysia Pahang
Pahang, Malaysia
khalidwsn15@gmail.com

***Abstract*— Convolutional Neural Networks (CNNs) are used for tasks such as object recognition. Once a CNN model is used in a radiative environment, reliability of the system against soft errors is a crucial issue, especially in safety-critical and high-performance applications that bound with real-time response. Selectively-hardening techniques do improve the reliability of these systems. However, the hard question in selective techniques is "how to exclusively select code portions to harden, to safe the performance from being degraded". In this paper, we propose a comprehensive analysis methodology for CNN-based classification models to confidently determine the only vulnerable parts of the source code. To achieve this, we propose a technique, Layer Vulnerability Factor (LVF) and adopt another technique, Kernel Vulnerability Factor (KVF). We apply these techniques to GoogLeNet, which is a famous image classification model, to validate our methodology. We precisely identify the parts of the GoogLeNet model that need to be hardened instead of using expensive duplication solutions.**

Keywords*— *convolutional neural networks, GoogLeNet, reliability, soft errors, GPUs

I. INTRODUCTION

Convolutional Neural Networks (CNNs are the new revolution that dominating the Artificial Intelligence (AI), and they are the most efficient technique to perform computer vison tasks, such as image classification [1] and object detection [2]. Due to the computational requirements, CNNs are often run on heterogeneous systems that composed of CPUs and accelerators, such as Graphics Processing Unit (GPUs). In fact, GPUs nowadays are the dominated hardware to accelerate CNN models [3].

CNN models are widely used in safety-critical systems, such as self-driving cars [4] and space applications [5]. Therefore, analyzing the reliability of such systems is critical. One way to address reliability issues is utilizing software redundancy. A bunch of techniques have been proposed for soft errors mitigation in GPUs based on software solutions, including Double Modular Redundancy (DMR), Triple Modular Redundancy (TMR), and Algorithm-Based Fault Tolerance (ABFT). However, the major challenging of using these techniques is the runtime overheads that associated with these solutions.

In this study, "which portion of the source code to harden" is the question we try to answer, and implement it on GoogLeNet algorithm as a case study. To answer this question, we propose a systematic analysis methodology for classification models to identify code portions that worth protecting. As a first objective, we propose a technique, Layer Vulnerability Factor (LVF) and adopt an exist technique, Kernel Vulnerability Factor (KVF). We implement these techniques on a CNN model that is used in image classification tasks, GoogLeNet. Using our methodology, we are able identify different vulnerable parts of the GoogLeNet model that need to be hardened.

The main contributions of this work are: (1) a methodology to evaluate the likelihood of faults in specific parts of the source code that likely to cause errors at the output; (2) the LVF concept and implementing it to a realistic case-study; and (3) an extensive analysis of GoogLeNet characteristics under SASSIFI fault injection.

The remainder of the paper is organized as follows. Section II shows the background and reviews related work. Section III presents the proposed methodology. Section IV analyzes and discusses the results. Section V concludes the paper.

II. BACKGROUD AND RELATED WORK

In this section, we present a brief background on Convolutional neural networks and especially GoogLeNet. We then, summarize previous findings on CNNs reliability.

a. Convolutional Neural Networks

Due to their outstanding performance that bypassed even the human ability in object recognition benchmarks (i.e., classification), CNNs are arguably the most popular type of the Deep Learning architectures [6]. The convolutional operation is the key component in CNNs. They use filters to extract features of the image, by sliding a filter over the input image, multiplying and accumulating products at every position of the input (i.e., receptive field) with this filter [7]. The well-known CNN architectures include AlexNet, VGGNet, GoogLeNet, ResNet, and DenseNet.

b. GoogLeNet

GoogLeNet developed by Google, is the winner of the ILSVRC (ImageNet Large Scale Visual Recognition Challenge) in 2014. It is the first CNN architecture replaced the expensive Fully-connected layers at the end of the model with a simple global average-pooling layer, which averages out the given values of each feature map. This change has dramatically reduced the number of parameters used in the model, which made it a faster in the training phase, lighter in size, and higher in performance, compared to its predecessor architectures, such VGGNet and AlexNet [8]. For these reasons, GoogLeNet has been widely adopted in many applications, including self-driving cars [9].
GoogLeNet handles our input data (i.e., images) in a fixed stack of operations to give the desired predictions. The image begins from the first layer (input layer) passes across

978-1-7281-2680-7/20/$31.00 ©2020 IEEE

multiple layers until reaches the output layer as output of the model. The main structure is as follows:

Layers: GoogLeNet's structure is built around several operations (i.e., layers). Here we briefly describe these layers: **Convolutional layer (Conv.)** is the main powerhouse of all CNNs. It performs dot-product operation for two matrices (i.e., input/image and a filter) to extract features from the input patterns. **Activation layer (Activ.)** is an activation function for adding non-linearity to the Conv. layers. **Normalization layer (Norm.)** is an important technique for improving the performance. It is used to normalize the inputs by adjusting and scaling them. **Max-pooling layer (Max.)** for down-sampling the feature maps, as its name suggests, it takes the maximum from the window. It is used to overcome overfitting problem. **Average-pooling layer (Avg.)** prepares the model for the final classification. Finally, **Softmax layer** is the classifier of the model.

The basic process of how GoogLeNet model works is this: First, the input (i.e., image) is passed to the first *Conv.* layer, where filters are used with different sizes (3x3 and 1x1) to generate feature maps. Then, this feature map will be passed to the *Norm.* layer and *Activ.* layer subsequently. Next, this the obtained output is passed to a *Max.* layer. It is worth mentioning that every *Conv.* layer is followed by *Norm.* and then *Activ.* layer, whereas only some *Conv.* layers are followed by a *Max.* layer. This block is repeated several times until the process reaches *Avg.* layer. Finally, *Softmax* layer to solve the multi-class classification problem. It is a probabilities vector of a thousand classes [8].

Our goal is to analyze and evaluate the vulnerability and resilience of each layer, to distinguish between reliable and vulnerable layers to soft errors.

Kernels: Kernel is the function (portion of the source code) that is executed on GPUs. It is important to highlight that a single CNN layer, such as Conv. and Norm., can be composed of several kernels. Usually, more than one kernel is invoked while executing the same layer. This means that a bunch of kernels directly contribute to the weakness or robustness of a certain layer. Due to the fact that each kernel has its own computational characteristics, we present a brief disruption of all the static kernels that are required by GoogLeNet model. It is worth mentioning that as our study is performed on pre-trained models, only kernels that are used for inference will be considered, training kernels are not included (see Table 1). Kernels in Table 1 based on Caffe framework that has been adopted on Darknet [10], which we use in our study. Ten kernels are used to make the required predications. Fill_gpu kernel, for filling the GPU buffer with image data (our input), weights (filters) and biases; and that is before performing any computations. Copy_gpu kernel, for Norm. layers to act just like Fill_gpu for them. Im2col_gpu kernel it does the image-to-column transformation before performing the matrix multiplication (MM) for Conv. layers.

TABLE I: GOOGLENET INFERENCE LAYERS AND KERNELS WITHIN EACH LAYER

Layer	Kernel
Conv.	Fill_ gpu, Im2col_gpu, *Add_bias.*
Max.	Forward_maxpool_gpu, Fill_gpu
Norm.	Copy_gpu, Normalize_gpu, Scale_bias, *Add_bias*
Activ.	Activation_array
Avg.	Forward_avgpool_gpu, Fill_gpu
Softmax	Softmax, Fill_gpu

Therefore, it is in charge of preparing a 2D array out of 3D array image to be treated as a matrix. ***Add_bias*** kernel, as the name suggests, to add biases to the feature maps after *MM* operations. **Scale_bias** kernel, to divide input values by their standard deviation in order to have a variance of approximately one. It is for *Norm.* layer. ***Normalize*_gpu** kernel, to achieve *Norm.* layer's purpose as defined as it is, to normalize the inputs to GPU. **Activate_array** kernel, to add nonlinearity feature to the feature maps before the next layer. **Maxpool** kernel, for removing the unnecessary information from the feature maps before the next *Conv.* layer. Finally, **Avgpool** and ***Softmax*** kernels perform the exact tasks that defined in their corresponding layers, *Avg.* and *Softmax* layers.

Our goal here, therefore, to further analyze and evaluate the vulnerability and resilience of each kernel within each layer, to identify the vulnerable kernels that should be selectively hardened instead of the whole layer. to minimize the associated overheads.

c. Related Work

Previous studies [7, 11] have evaluated the reliability of CNN models on GPUs. However, other architectures such as ResNet for image classification and YOLO for object detection have been examined. L. Weigel *et al.* analyzed the reliability of a feature descriptor for object detection called Histogram of Oriented Gradients (HOG) implemented on a GPU [11]. Despite this work introduced the concept of Kernel Vulnerability Factor (KVF) that we adopt in our study, this technique is completely application dependent, which means that HOG's KVF evaluation cannot be applied to GoogLeNet architecture. Besides we study the characteristics of a different GPU application (i.e., CNN architecture), we propose another technique to establish a comprehensive analysis methodology. Additionally, as this work studied errors resilience of HOG, it has no layers in it. Whereas we study layer vulnerability as well, this leads to determining the most vulnerable layers of the model.

A recent work by Santos *et al.* [7], has done a wide range study on this topic. Nevertheless, we still can point out the specific research gap. This work analyzed three CNN architectures, namely YOLO, Fast R-CNN and ResNet. The First two models are object detection frameworks, while ResNet is an object classification model. Authors of this work demonstrated that, they injected faults into YOLO only. Thus, the other two models have been evaluated through beam experiment without injecting faults, which means that classification model has not been injected. In addition to considering the difference between detection and

classification models, this work examined YOLO framework from layer perspective and error propagation, but without mentioning kernels within each layer. While in our work, the evaluation is based on two-phased analysis (i.e., layers and kernels). For instance, in our work, we show that a kernel within normalization layers is one of the most vulnerable kernels, while these layers do not even exist in YOLO framework. More importantly, for soft error mitigation techniques, Santos *et al.* have suggested Algorithm-Based Fault Tolerance (ABFT), while our study suggests selective hardening solutions. This confirms the fundamental difference between these two works. Finally, from a selective hardening perspective, software fault injection is an unavoidable tool for the target algorithm as an analysis step to first identify the critical portions of the code. Therefore, to the best of our knowledge, this is the first study that characterizes the reliability of CNN models from two perspectives (layers and kernels within each layer). In our study, we perform a comprehensive analysis of GoogLeNet as a case study to implement our methodology.

III. FAULT-INJECTION METHODOLOGY

In this study, we aim to evaluate the reliability of GoogLeNet by performing a wide fault-injection campaign into the GPU that executes this model. Therefore, our objective is to identify the vulnerable portions of the model, more specifically, kernels or layers.

a. Fault-injection Setup

To achieve this purpose, we use a fault injector called SASSIFI [12], designed by NVIDIA. SASSIFI provides three different levels of injections: (1) Register File (RF) mode, to study the probability that a particle strike in register file produces an error; (2) Instruction Output Address (IOA) mode, to study the probability that the address of an executing instruction is subjected to errors; (3) Instruction Output Value (IOV) mode, to study the probability that the value of an executing instruction is subjected to errors [12]. Notice that, in both IOA and IOV modes, the target is the instruction output, the only difference is whether the address or the value is injected. For a thorough analysis, we use all of the three modes in our study.

SASSIFI allows as to perform several bit-flip models (BFMs). As it has been proven in previous studies that the *single* BFM is realistic for the RF mode [12], and *random value* represents all other BFMs [11]. We would like to declare that the chosen BFMs are single and random BFM for our analysis. Once the setup is ready, we inject 1000 injections at each mode (i.e., RF, IOA, and IOV). We then, gather the results and compare them with the error-free result of the model.

b. Layer and Kernel Vulnerability

To achieve our goals, which specified at the end of Section II-B, we propose a technique called Layer Vulnerability Factor (LVF), to measure the vulnerability of GoogLeNet layers. And we also adopt another technique introduced in a previous work [11], called Kernel Vulnerability Factor (KVF), to measure the vulnerability of kernels that struct the layers of GoogLeNet model. LVF and KVF are basically the probability of a fault in a layer or kernel to impact model's computations, respectively.

The IVF and KVF are completely architecture-dependent strategies. This means that, we can identify the vulnerable parts of any CNN architecture using these techniques, however, the obtain values IVF and KVF for GoogLeNet model cannot be generalized to other CNN architectures. Nevertheless, the analysis methodology we propose is likely to be valid for other classification models.

IV. RESULTS AND ANALYSIS

Following the methodology discussed in Section III. In this section, we present our obtained results and analyze them based on our methodology. To present a detailed analysis for GoogLeNet model, we perform model's sensitivity analyses by evaluating layers and kernels vulnerability perspectives.

a. Layer Vulnerability Analysis

In this subsection, we evaluate the vulnerability of layers in our model, and that is by calculating the LVF value of each layer. LVF is calculated by dividing the number of observed errors (i.e., SDCs, DUEs, and Masked) in that specific layer by the total number of injected errors. Fig. 1, x2, and x3 show the obtained LVF for SASSIFI's three modes. As demonstrated in Section II-B, GoogLeNet consists of 27 layers in total, 21 *Conv.* layers, 4 *Max.* layers (1M, 3M, 8M, and 19M), one *Avg.* layer (25Avg), and *Softmax* layer (26S) (see Fig. 1, 2, and 3).

Once faults are injected, we begin our investigation by comparing the golden output (error-free output) of the GoogLeNet with the one that we injected using SASSIFI. In each of the given layers (27 layers), we measure the percentage of DUE, SDC, and Masked errors. LVF is the unit for that measurement. As shown in Fig. 1, 2, and 3, earlier layers generate more errors than later ones, as we go deeper the error rate is gradually reduced. This is due to the fact that, as we get closer to the output, the input size is reduced due to *Max.* layers that halve the input size, which reduces the probability of the particle strike as well. Moreover, we should notice that input size and (number and size) of the filters have direct impact on layer vulnerability. The more filters with bigger size, the more errors will be generated, as in layers L0, L2, L7, and L18. In contrast, *Conv.* layers situated right after *Max.* layers including L4, L9, and L20 except L2, are likely to produce less errors. For the same reason, where L4, L9 and L20 all have 1x1 filter size and the number of filters in each is the half of the filters number in their preceded *Conv.* layers. Whereas in L2, the filter size is 3x3 and number of filters is 3 times of L0's (192 filters in L2 and 64 filters in L0). This justifies why L0 seems the most vulnerable layer.

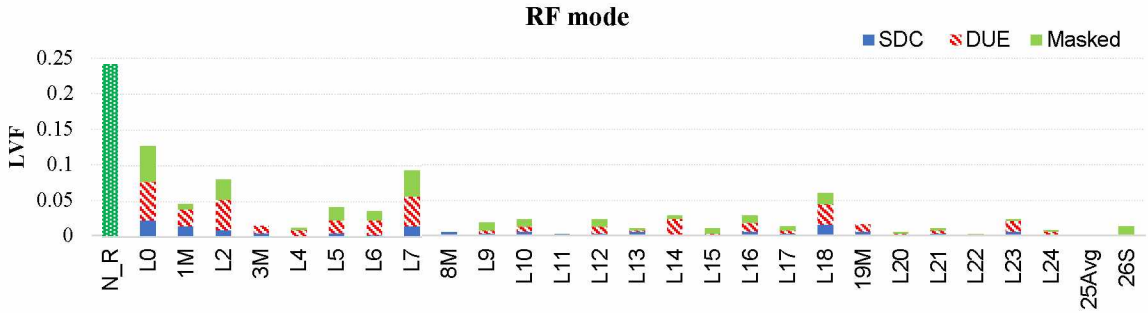

Fig. 1. LVF for each layer of GoogLeNet model, the result has obtained by injecting faults into the register file of the GPU

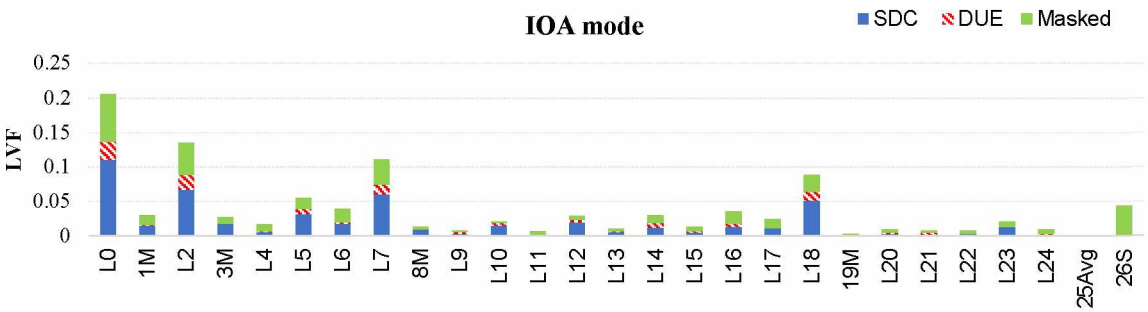

Fig. 2. LVF for each layer of GoogLeNet model, the result has obtained by injecting faults into *addresses* of the instruction outputs.

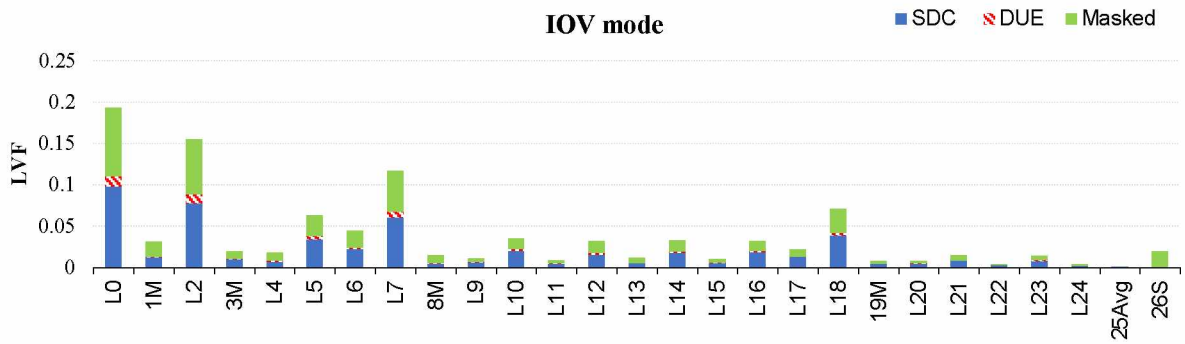

Fig. 3. LVF for each layer of GoogLeNet model, the result has obtained by injecting faults into *vaules* of the instruction outputs.

In RF mode (Fig. 1), an especial type of Masked errors called never read (N-R), at the very left of the graph. These are the errors that produced by our injection campaign but were never used by model's computations. That is the reason they do not belong to any layer. About 58% LVF of the Masked errors generated by RF mode were N_R errors.

Injecting faults into RF of the GPU tends to generate more DUE errors than injecting at the output of the destination register of the executing instructions (value or address). The justification for this phenomenon is that since the injections with RF mode (Fig. 1) will be in register index and stores address, RF injections are performed at a very lower level, and thus, it reasonable to stop the program execution in a form of hang or crash (i.e., DUE). While in IOA and IOV modes (Fig. 2, and 3), as they change the value or address of an executing instruction, they often end up with a mistake in the calculations that may lead a wrong output, rather than terminating the application, and few times, such as "illegal memory access" can cause DUEs.

We find Max. layers are interesting. Although they produce insignificant amount of errors (in the three injection modes), only 1M masked any of the injected faults in RF mode (see Fig. 1). While in IOA and IOV modes, at least 40% LVF of the injected faults are masked in each Max. layer. We believe that since Max. layers have no weights and only one math operation involved (i.e., taking the biggest number to remove the unnecessary information), some of the N_R errors should belong to Max. layers in RF injections (see Fig. 1).

In instruction output cases, however, even that single operation could be impact, that is the reason we have few DUE and SDC errors. In general, as *Conv.* layers form 78 % (21 out of 27 layers) of the model's structure, it is not surprising that the vast majority of the errors occurred at *Conv.* layers. However, since they dynamically invoke the

same number of static kernels, we can go further and identify all the kernels that contribute to the vulnerability of *Conv.* layers. This is what we present in Section IV-B.

Softmax and Avg. seem to be the most reliable layers in all injection modes. Besides, both of them are invoked only once, their kernels only spend short execution times on the GPU resources, which comes from the fact that few threads will be active. Further, Softmax layer masked all the faults injected into it (see Fig. 1, 2 and 3). This finding is significant, it tells us how to avoid hardening unnecessary portions of our algorithm, to save the performance of model from degradation. It is worth noting that although every *Conv.* layer is followed by *Norm.* and *Activ.* layers, they are not independent layers. Nevertheless, they have impact on model's reliability, especially *Norm.* layers. In Section IV-B, we analyze these two layers from their kernel's perspective.

To conclude it, the various layers of GoogLeNet have different sensitivity to errors. With our first evaluation metric (LVF), we are able to identify the four most susceptible layers (**L0**, **L2**, **L7**, and **L18**) out of 27 layers. However, based on our methodology, the decision of which portion of the code should be hardened requires kernels' evaluation as well, to exclusively harden the susceptible part of the layer.

b. Kernel Vulnerability Analysis

Layers are composed of smaller pieces of code, *kernels*. Therefore, in this subsection, we further investigate the vulnerability of our model through its kernels within each layer. For each kernel, we measure the KVF value, which is calculated by the same approach we followed in layers, but the calculations this time will be per kernel in each layer.

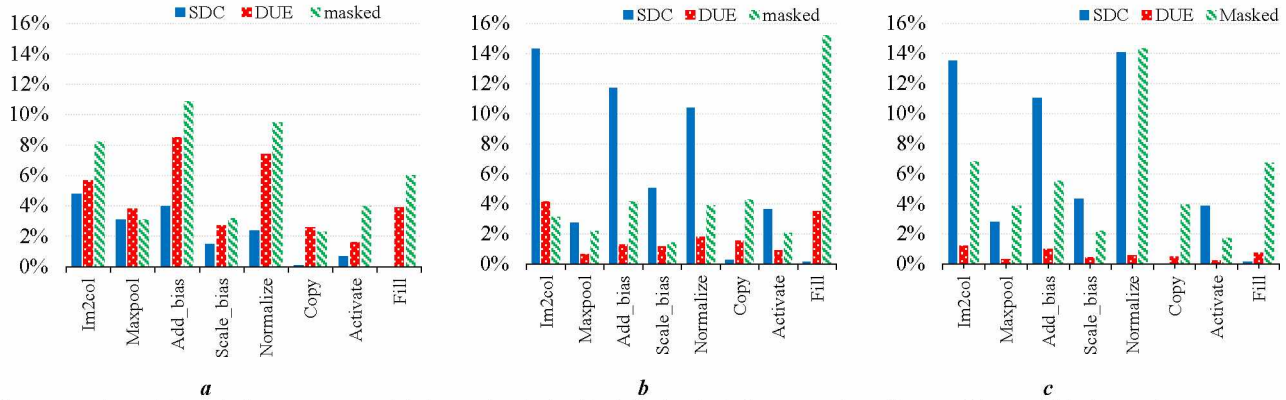

Fig. 4. KVF for each kernel of GoogLeNet model, the results obtained by injecting faults into **(a)** register file, **(b)** addresses of the instruction outputs, **(c)** values of the instruction outputs

Fig. 4 shows the KVF values in the three SASSIFI's modes: RF (Fig. 4a), IOA (Fig. 4b), and IOV (Fig. 4c). Fig. 4a shows that faults injected on *Im2col* kernel using RF mode produced 4.9% KVF of SDCs and 5.7% KVF of DUEs. While the same kernel using IOA and IOV injections produced 14.3% and 13.5 KVF of SDCs s, and only 4% and 1% KVF of SDCs in Fig. 4b and Fig. 4c, respectively. *Im2col* kernel is susceptible against RF faults and extremely susceptible to instruction output injections. As explained in Section IV-A, since IOA and IOV change the address and value of the currently executing instruction, respectively, it is reasonable that changing in destination register significantly affects the final output more than terminating the application. Considering the three injection modes, we found that *Im2col* is the most vulnerable kernel. The direct reason as we believe is that, besides it is invoked many times by *Conv.* layers, this kernel also uses many threads per a call (i.e., transforming every single receptive field of every filter into a column).

Add_bias and *Normalize* kernels have almost the same sensitivity against soft errors. Where injected faults on each of which, using RF mode, at most generated SDCs and DUEs 4% and 8% KVF, respectively. Whereas injecting faults using IOA and IOV on the same kernels produced the same percentage of DUEs but SDC percentages have small variety, 11.7% and 11% KVF for *Add_bias*, 10.4% and 14% KVF for *Normalize* in IOA and IOV, respectively. It is obvious that these two kernels come in the place of the most prone kernels to soft errors. It is important remembering that *Add_bias* is only invoked by *Norm.* layer unless the network does include it, then it is invoked by *Conv.* layer. Since *Norm.* layer always follows *Conv.* layer in execution, it is not surprising that *Add_bias* and *Normalize* kernels have nearly the impact of *Im2col* due to the exact number of invocations and also the nature of jobs they perform as explained in Section II-B. *Fill* and *Copy*, on the other hand, appear to be quite reliable kernels. Depside they produce a trivial number of DUEs, they either mask all SDCs or produce less than 0.02% KVF. Thus, they must be avoided from being hardening.

V. CONCLUSION

As it allows assessing the trade-off between the utilization of mitigation techniques and the associated runtime overheads, precisely evaluating soft error rate of a CNN model executed on modern GPUs is a crucial step. In this paper, we have proposed a systematic analysis methodology for CNN-based models to confidently identify the portions of the given algorithm that worth hardening for hardening efficiency. we proposed a technique (LVF) and adopted another one (KVF) to achieve our proposed methodology. We have performed an in-depth analysis of GoogLeNet, a well-known CNN

architecture. We showed that only fewer layers are vulnerable. And by understanding the error propagation within each layer, we found that only tiny portions of the GoogLeNet algorithm are susceptible to soft errors, which are three kernels, namely: *Im2col*, *Normalize*, and *Add_bias*. Finally, the presented methodology likely to be validate for other CNN-based classification algorithms, to be used as a pre-step for selective hardening techniques in order to avoid unnecessary expenses of duplication solutions.

ACKNOWLEDGMENT

This work is supported through Innovation Foundation of Radiation Application, China Institute of Atomic Energy (KFZC2018040205). It is also supported by Changzhou Sci & Tech Program (Grant no. CZ20180003).

REFERENCES

[1] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 3642-3649, 2012.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", In NIPS, pp. 91–99, 2015.

[3] Z. You et al., "White Paper on AI Chip Technologies", Beijing Inno. Cen. Fut. Chips (ICFC), Beijing, China. 2018.

[4] K. V. Sakhare, T. Tewari, and V. Vyas, "Review of Vehicle Detection Systems in Advanced Driver Assistant Systems," Springer. https://doi.org/10.1007/s11831-019-09321-3, 2019.

[5] S. Rawat, "Airplanes Detection for Satellite using Faster RCNN," Medium, https:// towardsdatascience.com/airplanes-detection-for-satellite-using-faster-rcnn-d307d58353f1.

[6] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, 2017.

[7] F. F. D. Santos et al., "Analyzing and Increasing the Reliability of Convolutional Neural Networks on GPUs," IEEE T Reliab, vol. 68, no. 2, pp. 663-677, 2019.

[8] C. Szegedy et al., "Going deeper with convolutions," in 2015 IEEE Conference on (CVPR), pp. 1-9, 2015.

[9] M. Al-Qizwini, I. Barjasteh, H. Al-Qassab, and H. Radha, "Deep learning algorithm for autonomous driving using GoogLeNet," in 2017 IEEE Intelligent Vehicles Symposium (IV), pp. 89-96, 2017.

[10] J. Redmon, "Darknet: Open Source Neural Networks in C," [online]. Available: http://pjreddie.com/darknet/, 2016.

[11] L. Weigel, F. Fernandes, P. Navaux, and P. Rech, "Kernel vulnerability factor and efficient hardening for histogram of oriented gradients," in 2017 IEEE Inter. Symp Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1-6, 2017.

[12] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer, "Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation," Intern. Sym Performance Analysis of Sys and Software (ISPASS), California, USA, 2017.