POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

A methodology for the reliability analysis and the efficient hardening of Convolutional Neural Networks

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Alessandro Nazzari

Advisor: Prof. Antonio Rosario Miele

Co-advisor: Prof. Luca Cassano

Academic year: 2021-2022

1. Introduction

Convolutional Neural Networks (CNNs) usage has been steadily increasing in the last decade, especially for perception functionalities in both safety-critical systems and not. An example of this phenomenon is the Autonomous Driving System (ADS), a set of high-level functionalities aiming at partially or entirely substituting the human driver. Design principles for digital systems in safety-critical applications are strictly regulated by standards such as ISO 26262 [5]. Similarly, the Society of Automotive Engineers (SAE) regulates ADS functionalities. A high level of reliability and fault detection mechanisms are required by both standards. Overall the request for the reliability of digital systems has always been crucial.

In 1996 Boeing Defence and Space Group published a research paper [6] stating that the occurrence of faults in ground segment systems averages around two every thousand billion hours. While it seems a negligible rate, the number of cars used has exponentially increased since 1996, with an average of 268 million vehicles used in Europe in 2019. Considering this number, the estimate of faults per car would be one every 3.7 hours, concerning information indeed. In this scenario, classical redundancy approaches may not be feasible due to the high complexity of the models and the hardware used. However, deep learning applications have an intrinsic degree of fault resiliency. Many tools can be used to analyze the reliability of CNNs. We decided to adopt and improve an existing Error Simulator called CLASSES.

2. Goals and Contribution

The goals of this thesis were threefold:

Improve CLASSES: The first one was to improve the prototype implementation of the error simulator framework. It was required to identify the most critical components of the framework and design solutions aimed at fixing them. Moreover, we improved the usability of the tool and designed a more flexible interface allowing for easy usage of all its components.

Extensive error simulation campaign: The second goal was to design and execute an extensive error simulation campaign. The objectives of this campaign were, in turn, twofold. Firstly to validate the framework's effectiveness and demonstrate that it is possible to perform an error simulation campaign of substantial size. Secondly, to assess the robustness against faults

of multiple CNNs to verify that the results obtained with CLASSES are meaningful.
Hardening of CNNs: The third and final goal of the thesis was to define a hardening strategy for CNNs capable of exploiting in an automated and effective way information produced by CLASSES.

3. Background

3.1. Convolutional Neural Networks

Convolutional Neural Networks are a particular class of Neural Networks that presents superior performances with image, speech, or audio signals. They manage multidimensional data, known as tensors, and aim at deriving semantic representation from the input to accomplish a high-end task. CNNs are internally organized in a sequence of layers implementing different operators, mainly Convolutional, Pooling and Fully-connected ones. Each model is composed of various blocks containing those main ones and other smaller layers.

3.2. Reliability Analysis Tools

Reliability analysis of machine learning models has always been a fundamental topic among researchers. Several tools and solutions have been developed to carry on this task. At the application level, one of the most used groups of tools consists of error simulators. As the name suggests, they replicate the presence of an error in the data path of the program. Their operation is explained in Section 3.3. Among the existing tools, the two most interesting ones are CLASSES [1] and TensorFI [2]. CLASSES is an error simulator framework with the particularity that the error models available are generated from an error injection campaign and are thus validated against real-world observations. Unlike a fault injector, an error simulator works at the application level and introduces errors based on a database of models. It allows the developer to observe the faults' effects on the application during the design phase instead of waiting for the deployment of the completed model. TensorFI is also an error simulator framework that presents a limited set of fault types and allows the user to specify which class of operator to target.

3.3. CLASSES

We decided to use CLASSES due to its flexibility, ease of usage and methodological robustness in presenting validated error models. The framework is composed of two sections:
Error model generation phase. It is initially run on a target GPU where custom scripts execute a set of selected ML operators. An architectural error injection fault is used during the execution of these scripts, and their outcome is compared with a golden version. The faulty outputs are analyzed to generate models that provide a structured description of the observed errors.
Error simulation phase. In this second phase, the user selects a specific layer of a CNN then the tool extracts an error model from the database. The error model and its probability of being selected are based on what was observed in the first half of the framework. The extracted error is injected into the chosen layer allowing the developer to analyze its effects on the model's outcome.

4. Improving CLASSES

While using the framework, we identified some problems related to the technologies used. This section explains the issues and how we decided to deal with them.
Architectural Fault Injector. The first implementation of CLASSES integrated SASSIFI [4]. In this work we introduced NVBitFI [7] which is the state-of-the-art fault injector. The main advantage of this tool is that it can inject any CUDA binary without needing the source code. Moreover, it is capable of targeting dynamically loaded libraries which is a crucial improvement over SASSIFI. Moreover, we developed an automating script for the execution of the fault injection campaign.
Caffe. CLASSES relied on Caffe to implement test applications for each CNN operator to be used in fault injection campaign for error modeling. We switched to cuDNN for multiple reasons:

- ***Better support:*** cuDNN has a better online support than Caffe which has been discontinued.
- ***Used by TensorFlow:*** TensorFlow's back-end is built on top of cuDNN. Using

this library ensures that the algorithms' implementations are perfectly matched.
- ***Easier to use:*** tensors extracted from TensorFlow can be directly used by cuDNN while they had to be reshaped in order to be used in Caffe.

To move from Caffe to cuDNN, we were required to write the code that executes each operator from scratch. Due to the initial complexity of this process, we developed a skeleton program to explain all the steps required with clarity and made it available in the framework for anyone interested in using it.

Support for different ML frameworks. We tested the portability of our approach by designing a Keras implementation of the Error Simulator developed as a custom layer that can be inserted into a model. We also developed a script to correctly upload the saved weights for the model after the introduction of said layer.

Enhancing CLASSES' usability. In addition to the technological improvements we also increased the tool's usability and made it more flexible. We wrote an extensive guide on how to use the tool and improved the input/output interfaces allowing an easier usage of the framework.

5. Case studies in reliability analysis

The two objectives of this phase were to validate the framework's effectiveness and assess the robustness of multiple CNNs. To do so, we selected four different models belonging to either the Image Recognition or the Steering Angle Detection class of networks. We selected these two domains due to their importance in the considered working scenario. Moreover, to test the tool against the maximum number of conditions, the four models have different depths, and we trained them against three distinct datasets. They are CIFAR10, the German Traffic Sign Recognition Benchmark and a custom set of images taken from a moving vehicle.

For an exhaustive execution of the campaign, we ran the error simulator approximately five thousand times for each tuple image and targeted layer.

To correctly analyze the results, CLASSES needs an oracle function that classifies a single outcome as usable or not. We decided that a result is considered usable when dealing with Image Recognition models if the produced label is the same as the expected one. Differently, in the Steering Angle Detection, we accepted all the outcomes where the absolute value difference between the nominal result and the faulty one is less than five degrees. The choice of the oracle is entirely context-dependent, and it can be changed at any time. An example would be to check the absolute difference between probabilities instead of predicted labels. It does not influence the methodology's validity.

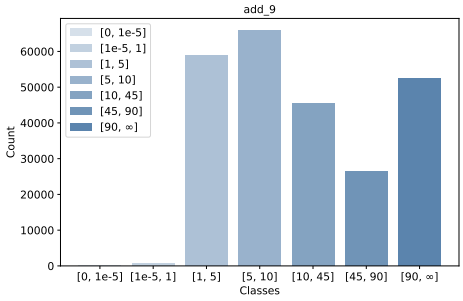

Figure 1: PilotNet Add9 Layer

5.1. Results

Among the results obtained we can highlight some of the most interesting:

Different layers behave differently: The operation that a layer performs can be either complex, i.e., convolution, or simple, i.e., add. A complex operator generates error models with a higher number of corrupted values. The effect of this behavior is that, depending on the layer's type, the impact it has on the model's outcome is different. It can be understood by observing Figure 4.7 and Figure 4.8. Despite being nearly at the same depth in the net, their influence is vastly different, with the former having a distribution of differences spread among all the classes while the latter resides almost entirely in the $[90; \infty]$ range.

Closer to the output means stronger effects: A layer's impact on the outcome is affected by its distance. A fault striking the first operation of the model will have a lower effect on the output since there is a high number of layers that will recover its damages. Images 3 and 1 show two different add operators where the former is closer to the output and has a much stronger impact.

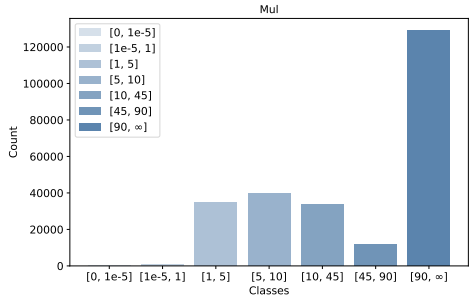

Figure 2: PilotNet Mul Layer

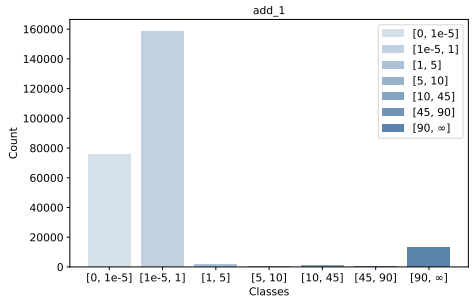

Figure 3: PilotNet Add1 Layer

Classes to look out for: Considering the image classification models, we analyzed the misclassifications. Figure 1 shows the distribution of errors among the various classes for the Cifar10 model. Classes 1, 4, and 9 have a much lower count of misclassifications than the others. It means that the model, despite being affected by faults, correctly labels images belonging to those classes with a much higher rate than the other classes. It means that, whenever the model predicts one of those classes, the developer can be sufficiently sure of the result without needing any other tools to verify.

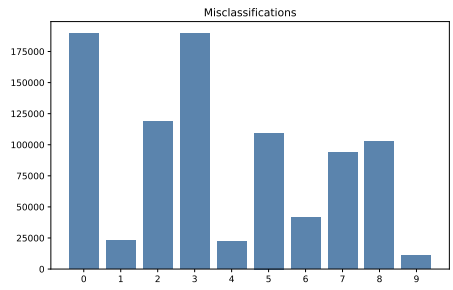

Figure 4: Cifar10 misclassifications distribution

6. Hardening CNNs

With the execution of the error simulation campaign, we were able to analyze the CNNs' critical issues at the layer level. We can now take advantage of these observations to define hardening methods.
The last goal of the thesis was to create a reliability analysis technique that uses the results of the error simulation campaign. The information's granularity was at the layer level and, for sake of simplicity, we assume that each layer is composed of a single operator.

6.1. CNN robustness

We designed an automatic way of calculating the robustness of a CNN based on CLASSES framework. We based our analysis on the concept of Layer Vulnerability Factor [3] which is defined as the probability that a fault striking a layer affects the outcome of the whole model.

$$LVF_i = \frac{\#sdc_i}{\#exps} \tag{1}$$

We then changed this metric to reflect the concept of usability by replacing $\#sdc_i$ which is the number of SDCs with $\#u_i$ which is the number of unusable results.
Operator's susceptibility: this metric quantifies how many times an error striking the operator under analysis results in a corrupted output or not

$$p_{err_i} = \frac{\#errors_i}{\#faults_i} \tag{2}$$

Overall robustness against operator i: this metric defines, for each operator, the effects it has on the model's robustness. The operator's susceptibility is scaled with the ratio between usable results and error simulations performed.

$$R_i = (1 - p_{err_i}) + p_{err_i} \cdot \frac{\#u_i}{\#sim\_errs_i} \tag{3}$$

Timing weight: the last metric considered is the ratio between an operator's execution time and the whole model's execution time.

$$T_i = \frac{\Delta t_i}{\Delta t_{CNN}} \tag{4}$$

We can combine Equation 3 and Equation 4 to produce the overall CNN robustness, defined as:

$$R_{CNN} = \sum_{i \in CNN} R_i \cdot T_i \tag{5}$$

6.2. Hardening Strategy

R_{CNN} measures the CNN's intrinsic robustness against faults, i.e., its capability to output usable values even in presence of faults. Defining a hardening technique that maximizes this metric while limiting the performance overhead is a goal of the thesis. We selected Duplication With Comparison (DWC) applied at the granularity of a single layer. It consists in executing two instances of the same layer and comparing the output with an ad-hoc checker. This scheme can detect errors in the single layer run. Duplicating a layer improves the CNN robustness while incurring in a performance degradation PD_i due to the additional execution of the layer. We propose to selectively harden the model by applying DWC to a set of layers leaving the others unprotected. This approach improves the overall robustness of the CNN while incurring in limited performance degradation. A duplicated layer is characterized by:

$$R_i = 1 \tag{6}$$

$$PD_i = t_i + t_{c\_i} \tag{7}$$

We adopted a multi-objective Design Space Exploration (DSE) process that analyzes all possible combinations of duplicated layers.
Each solution is characterized by a subset of hardened layers H_CNN leaving the others unprotected. The robustness of any given combination is thus calculated as:

$$R_{SH\_CNN} = \sum_{i \in \overline{H}\_CNN} R_i \cdot \frac{t_i}{t_{CNN}} + \sum_{i \in H\_CNN} \frac{t_i}{t_{CNN}} \tag{8}$$

$$t_{SH\_CNN} = t_{CNN} + \sum_{i \in H\_CNN} PD_i \tag{9}$$

It is then up to the designer to choose among those depending on the time constraint of the model.

6.3. Hardening results

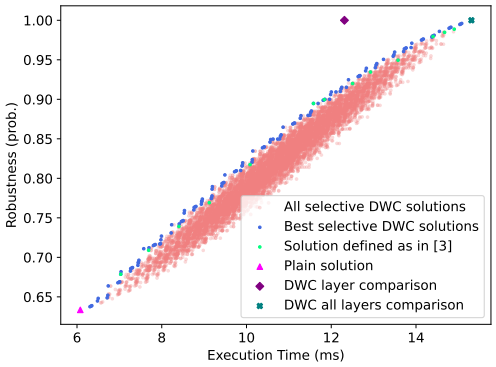

Figure 5: VGG11

Figure 5 shows the DSE for the VGG model. The blue points represent the Pareto front. These are the solutions for which the robustness is maximized under a certain time threshold. Red dots instead represents all the possible solution in the design space. We also identified two particular hardening schemes.
The first one is the DWC outcome comparison, in which we duplicate the whole model but apply the checker only at the end.
The second one is DWC all-layers comparison, where we duplicate all layers and check every result. Clearly, both schemes have perfect robustness. The main difference is the execution time.
In VGG, the time of each checker is comparable to the time of each layer, making these solutions more favorable than most of the cases explored. Instead, if we consider a model such as PilotNet, where the execution time of each layer is consistently higher than those of the checkers, both solutions are at the top-right of the Graph 6. They have the best robustness but also the highest execution time.
It is of particular interest to highlight the green dots in the graph. Those represent the hardened solutions as discussed in [3]. The proposed approach consists of ordering all the available layers based on the LVF. And then hardening them one by one, starting from the most critical one. This solution is faster since the number of cases to consider is much smaller but also less precise.

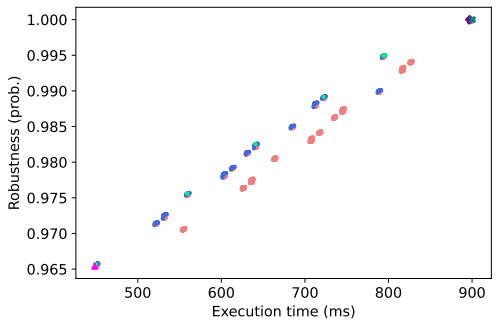

Figure 6: PilotNet

We can conclude our analysis with two more observations.

Some models are intrinsically robust. Looking at Figure 7, we see how the robustness of the base model is already really high, almost 0.93. It is an intrinsic property of this particular model due to how it was built, and it does not depend on the depth of the net itself. Looking at Figure 5, we have a deep model with much lower base robustness or, looking at Figure 6, the opposite, a model with a low number of layers and high base robustness. It means that, in order to design a reliable model, the developer must carefully choose how the net is constructed and then use our technique to improve the base result.

Great improvements might come at low costs. Figure 7 is a clear example of how improvements might not be expensive. Reaching maximum robustness of 1 would require an execution time of approximately 1200 ms. We can reduce the execution by nearly 200ms, almost 17%, while keeping the robustness over 0.98.

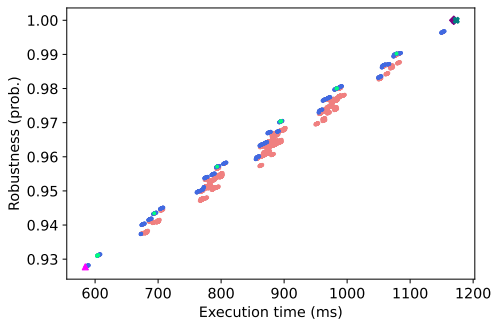

Figure 7: Cifar10

7. Conclusions

The first two goals of the thesis were to improve the existing error simulator and use it in a structured and comprehensive error simulation campaign. We were able to complete these goals. CLASSES is now accessible to anyone interested in using it, and it is easy to exploit its functionalities. The campaign we performed highlighted multiple interesting points on how CNNs react to the presence of faults and the effects they have on the outcomes. While these results alone are already relevant for any developer interested in hardening theirs ML model, we decided to move one step further and design an indicator that uses the information produced and uniquely defines the robustness of a given CNN. Using this metric, we proposed a hardening technique that, through DWC, aims at finding the solutions that maximize the robustness while maintaining the execution time under a given threshold. Designing this metric was the third and final goal of the thesis.

References

[1] C. Bolchini, L. Cassano, A. Miele, and A. Toschi. Fast and accurate error simulation for cnns against soft errors. *To appear in IEEE Transactions on Computers.*

[2] Z. Chen et al. TensorFI: A Flexible Fault Injection Framework for TensorFlow Applications. *arXiv:2004.01743 [cs, stat]*, Apr. 2020.

[3] F. F. dos Santos, L. Carro, and P. Rech. Kernel and layer vulnerability factor to evaluate object detection reliability in GPUs. *IET Computers & Digital Techniques*, 2019.

[4] S. K. S. Hari et al. SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation. 2017.

[5] ISO Central Secretary. Road vehicles – functional safety. Standard, International Organization for Standardization, 2018.

[6] E. Normand. Single event upset at ground level. *IEEE Transactions on Nuclear Science*, 43(6):2742–2750, Dec. 1996.

[7] T. Tsai et al. NVBitFI: Dynamic Fault Injection for GPUs. In *Proc. DSN*, pages 284–291, 2021.