Advances in Intelligent Systems Research, volume 174

Proceedings of the 8th Scientific Conference on Information Technologies for Intelligent Decision Making Support (ITIDS 2020)

Evaluation of the Impact of Random Computing Hardware Faults on the Performance of Convolutional Neural Networks

Emil Valiev*
Faculty of Informatics and Robotics
Ufa State Aviation Technical University
Ufa, Russia
valiev.usatu@gmail.com

Andrey Morozov
Institute of Industrial Automation and Software Engineering
University of Stuttgart
Stuttgart ,Germany
andrey.morozov@ias.uni-stuttgart.de

Michael Beyer
Institute of Automation
Technische Universität Dresden
Dresden, Germany
michael.beyer3@mailbox.tu-dresden.de

Nafisa Yusupova
Faculty of Informatics and Robotics
Ufa State Aviation Technical University
Ufa, Russia
yussupova@ugatu.ac.ru

Klaus Janschek
Institute of Automation
Technische Universität Dresden
Dresden, Germany
klaus.janschek@tu-dresden.de

***Abstract*— Artificial Intelligence (AI) rapidly spreads across high-tech industries and enters almost every safety-critical area such as automotive, aerospace, and medical industries. However, like any other software, AI-based applications are prone to random hardware faults such as a random bit flip in CPU, RAM, or network. Therefore, it is essential to understand how various hardware faults affect the performance and accuracy of AI applications. This paper provides a general description and particular conceptual and implementational features of our recently introduced Fault Injection (FI) framework InjectTF2. InjectTF2 is developed using the TensorFlow 2 API and allows the user to specify fault parameters and perform layer-wise fault injection into the TensorFlow 2 neural networks. It enables the automated injection of random bitflips. The paper describes the software architecture of the framework. The framework is open source and freely available on the GitHub. The application of InjectTF2 is demonstrated with extensive fault injection experiments on a Convolutional Neural Network (CNN) trained using the GTSRB dataset. The experiments' results show how random bitflips in the outputs of the CNNs layers affect the classification accuracy. Such results support not only numerical analysis of reliability and safety characteristics but also help to identify the most critical CNN layers for more robust and fault-tolerant design.**

Keywords—deep learning, fault injection, random hardware faults, automated fault injection, Convolutional Neural Network, neural network fault tolerance

I. INTRODUCTION

Neural networks have gained recognition due to increased computing power, big data availability, and new training algorithms. Besides that, companies like Tesla are bringing more and more people into contact with autonomous driving and other technologies based on deep-learning. As an important part of Artificial Intelligence (AI), computer vision is a major enabler for autonomous driving. Many recent papers address the applications of Convolutional Neural Networks (CNN) for autonomous driving tasks [1, 2]. TensorFlow [3] is the most popular framework for the development of neural networks. It has proven to be a stable and reliable software platform. Nevertheless, neural networks, like any software, are susceptible to common random faults of the computing hardware where they are deployed. These faults include common bit flips in the cache of a processor, in memory, or network. The bit flips can be caused by radiation, electromagnetic induction, or the lowering voltage. They might seriously affect the classification accuracy of the network. Safety-critical applications that utilize DL methods have to ensure their reliability according to industry defined standards. Therefore, investigating the effects of such soft errors is crucial to guarantee the correct operation of the application under the impact of such faults.

Contribution: This paper presents the particular conceptual and implementational details of a new error injection framework for TensorFlow 2 - InjectTF2. The framework allows the user to specify fault parameters and perform automated layer-wise fault injection into the TensorFlow 2 neural networks. The application of InjectTF2 is demonstrated with extensive fault injection experiments on a Convolutional Neural Network (CNN) trained using the GTSRB dataset. The experiments' results show how random bitflips in the outputs of the CNNs layers affect the classification accuracy. Such results support not only numerical analysis of reliability and safety characteristics but also help to identify the most critical CNN layers for more robust and fault-tolerant design.

The remainder of the paper is structured as follows. Section II discusses the related work. The architecture of InjectTF2 is presented in Section III. Section IV demonstrates our case study experimental setup. The achieved results of the case study analysis are shown and interpreted in Section V. Section VI concludes the paper.

II. Related Work

In this paper, we describe the second version of the fault injection framework developed for the TensorFlow v2. The InjectTF for the first version of TensorFlow was presented in [4]. The authors described the tool and reported the significant loss of the classification accuracy. As the case study, they used an open-source traffic sign classifier [5] that is based on the VGGNet [6]. The network has been trained on an augmented German Traffic Sign Recognition Benchmark (GTSRB) dataset [7] that is split into three subsets for training, testing, and validation. Injections were performed randomly to bit-flip faults in different operations of the network graph. In conclusion, the authors were able to demonstrate the decrease in NN accuracy. Furthermore, authors noted that the framework should be ported to the new version of TensorFlow 2 to ensure timeliness [2].

Besides that, another similar fault injection framework for TensorFlow 1 exists. It is TensorFI [8] developed by our colleagues from the University of British Columbia. The TensorFI is similar to the InjectTF and is also developed for the first version of TensorFlow.

III. Fault Injection Framework InjectTF2

In this section, we present our fault injection framework for TensorFlow 2 – InjectTF2. The framework was developed in Python 3 using the API of TensorFlow version 2.0. It was designed in a modular way that enables further extension. We used the layer-wise fault injection methodology, which means that the user can configure the fault injection parameters for each individual layer. A CNN with the GTSRB dataset [7] was used for the feasibility study.

A. Methodology

TensorFlow 2 was released in September 2019 [9]. Many things have been changed in the framework in comparison to the first version. This includes the internal representation of the neural network, API design, improvements in performance, etc. TensorFlow 2.0 is focused on simplicity and ease of use. First of all, it integrates Keras [10] for easy model building and eager execution. TensorFlow 2.0 provides a more robust model deployment in production on any platform. More powerful experimentation for research and the simplification of the API by cleaning up deprecated APIs and reducing duplication are also among the features of the new TensorFlow version. TensorFlow 2.0 supports backward compatibility. However, you need to avoid it in order to exploit the full advantages of the latest version.

In the first version of InjectTF, the authors employed the operation-wise fault injection method. The errors were injected into the outputs of separate operations of the neural network graph. In InjectTF2 we use the layer-wise method. This choice is based on the fact that TensorFlow 2 uses high-level API Keras. Keras allows users to build their networks from layers. In this way, the layer turns into the basic unit instead of an operation. Furthermore, the Keras API does not allow the user to change operations inside a layer.

B. Architecture

InjectTF2 is built on top of the standard TensorFlow 2 API. InjectTF2 requires the following three inputs: a configuration file, a trained neural network, and a testing dataset. Fig. 1 shows these inputs on the left side. The neural network under test should be provided as a HDF5 model. The dataset should be given in the standard TensorFlow dataset format. In TensorFlow 2, in most cases, the model is built as a sequence of individual layers. However, for some complex neural networks, the layers can be connected no in a series, but form an acyclic direct graph. As of now, the InjectTF2 framework supports only sequential models developed with the high-level Keras API.

C. Configuration

The configuration file specifies layers that are to be injected and the fault injection probabilities. Currently, only bit flips can be injected by the framework. However, it is possible to choose between flipping a random bit or a specific bit of a random value of a layer's output. A fragment of a configuration file is shown below. It specifies that we will inject errors into the output of the third layer. The type of the fault is specified as a bitflip in a random value of the layer's output. The faults will be injected stochastically with the probability 0.001. The configuration file is in a human-readable YAML format [11].

```
1:  inject_layer_number:
2:      3:
3:          probability: 0.001
4:          fault_type: BitFlip
5:          elements: random
6:  ...
```

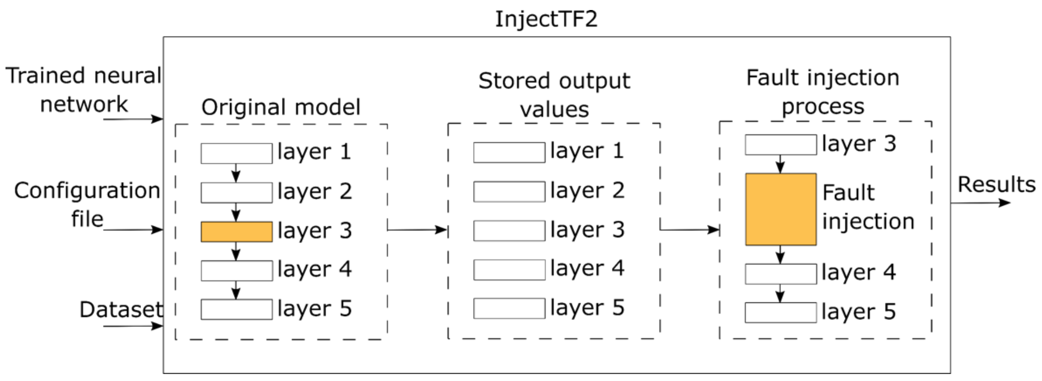

Fig. 1. Working principle of the fault injection framework InjectTF2.

D. *Working principle*

The working principle of InjectTF2 is shown in the middle of Fig. 1. During the initialization, we execute the model up to the layer where the errors are to be injected, using the provided data set. The output values of the injected layer are collected and stored. Afterward, we continue the analysis from the selected layer onward using the gathered values. Errors are injected into the stored values of the selected layer according to the parameters defined in the configuration file.

E. *Dynamic netwrok splitting principle*

Splitting the model and executing the two resulting parts separately drastically reduces the execution time of the experiments because the network is not executed from bottom to top each time. However, a substantial amount of memory is required to store the intermediate values of the selected layer. Depending on the layer and dataset size, this can quickly reach 100 GB.

F. *Software architecuture*

Like most high-level programming languages, Python supports object-oriented programming and allows the programmer to create classes. Thus, it logically encapsulates the functions associated with them in various modules. All classes and functions of the developed injection framework are shown in the UML Class Diagram Fig. 2. The project contains four classes in four separate modules. Each class/module is described in more detail below. The project is available in the GutHub: https://github.com/mbsa-tud/InjectTF2. There you can find more information about individual classes and functions.

InjectTF2 is the main class of the framework. It is basically the interface that provides the user with access to the full fault injection functionality of the framework. The work with InjectTF2 class starts with the *initialization* function. The neural network model, the path to the configuration file, the data set for network validation, and the batch size must be provided as parameters of this *initialization* function. However, the core function of the class is the *get_top_k_from_layer* function. It initiates the neural network and injects hardware errors according to the parameters specified in the configuration file. The *InjectTF2* module contains 156 lines of code.

ConfigurationManager is the class that implements the interaction with the configuration file. The function *_read_config* loads the configuration. The path to the file is an input parameter of this function. The *ConfigurationManager* class exploits the *yaml* library. The function *get_data* loads the configuration into the dictionary *_config_data*. The structure of the dictionary repeats the *yaml file* and is shown below.

```
"inject_layer_number": {
    "layer_number": {
            "probability": "",
            "fault_type": "",
            "elements": ""
    }
}
```

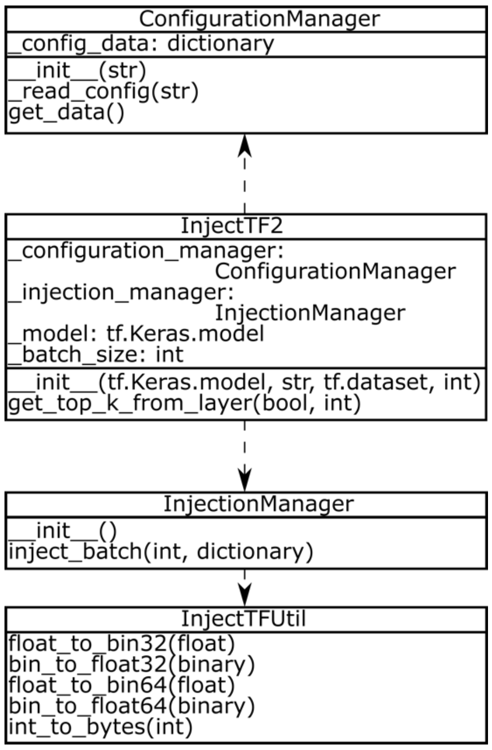

Fig. 2. UML Class Diagram of the InjectTF2 project.

Currently there is only *BitFlip* value for the *fault_type*. Allowed values for the probability are from *[0.0, 1.0]*. Allowed values for elements are from *{random, all}*. The number of lines of code of the *ConfigurationManager* equals to 52.

InjectionManager class is responsible for all activities related to the fault injection itself. During the injections of the faults into the provided neural network, this class generates the output of the injected layer based on the injection type according to the configuration file. The function *inject_batch* replaces the output values. The number values depend on the batch size and the layer with generated values with fault injections according to the configuration file. The number of lines of code is 131.

The other utility functions used by the framework are encapsulated in the *InjectTFUtil* module. For example, it contains functions that convert floating-point numbers to their binary representation and vice versa. These functions are used for the injection of bit flips. *InjectTFUtil* module contains just 42 lines of code.

IV. EXPERIMENTAL SETUP

Image classification experiments with Convolutional Neural Networks (CNN) have been conducted using the presented fault injection framework. For these experiments, we developed and trained a CNN, based on general design guidelines. It consists of 12 layers and uses the ReLU activation function for all convolutional and dense layers except the last one, which uses the Softmax function. The CNN structure is listed in Table 1. The first two columns of the table describe the consequent number and the type of the layer. The right-hand column of the table describes the output dimension of each layer. As it was discussed, the faults are injected into a random value of the layer output.

The CNN was trained using Google's Compute Engine on the augmented GTSRB dataset. It is capable of classifying traffic signs with an accuracy of approximately 96.4%. The accuracy improvement during the training process is shown in Fig. 3. We can se that an acceptable value accutrace is already reached after approximately 2.5 epoches. The resulted training accuracy is approx. 0.8.

TABLE I. THE STRUCTURE OF THE CNN UNDER TEST

Layer	Type	Output dimension
1	Convolutional	32×32×32
2	Convolutional	32×32×32
3	MaxPooling	16×16×32
4	Dropout	16×16×32
5	Convolutional	16×16×64
6	Convolutional	16×16×64
7	MaxPooling	8×8×64
8	Dropout	8×8×64
9	Flatten	4096
10	Dense	256
11	Dropout	256
12	Dense	43

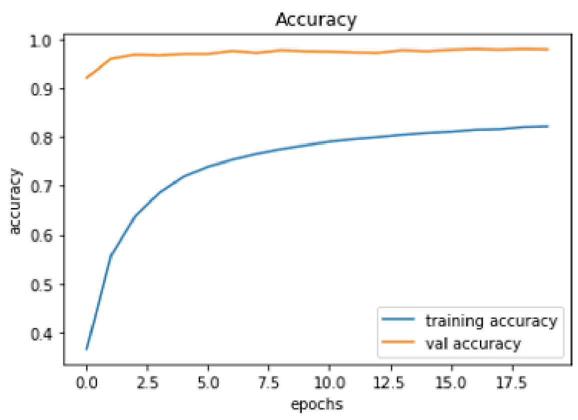

Fig. 3. The accuracy of the CNN during the training process.

The performance of the network is evaluated by calculating the classification accuracy using the GTSRB testing subset's ground truth. Each layer of the network is investigated in a set of separated experiments, whereby one bit of flip is injected with a varying probability into the respective layer's output. Probabilities vary from 5% to 50%, with step 5%. Depending on the probability of error injection, one random bit of one random element of the layer's output is flipped. Each of the selected probabilities is tested 100 times. The cumulative moving average (CMA) has been calculated for the resulting classification accuracy of each run of the experiments to verify that the sample size of the experiments is large enough, and the results are statistically confident.

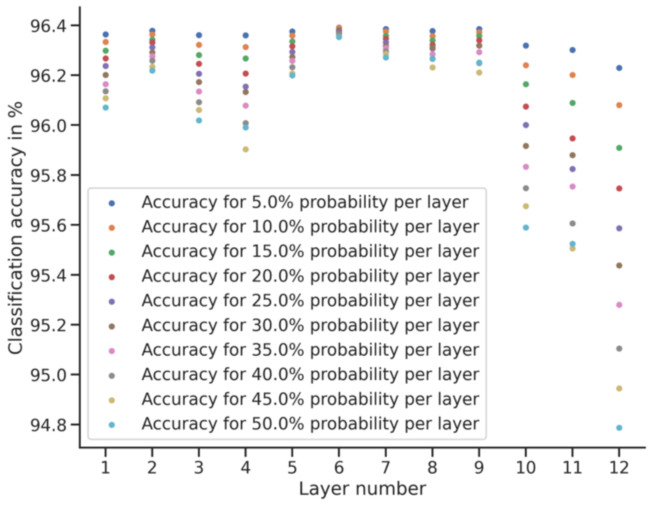

Fig. 4. Results of the experiment for varaying fault injection probabilities.

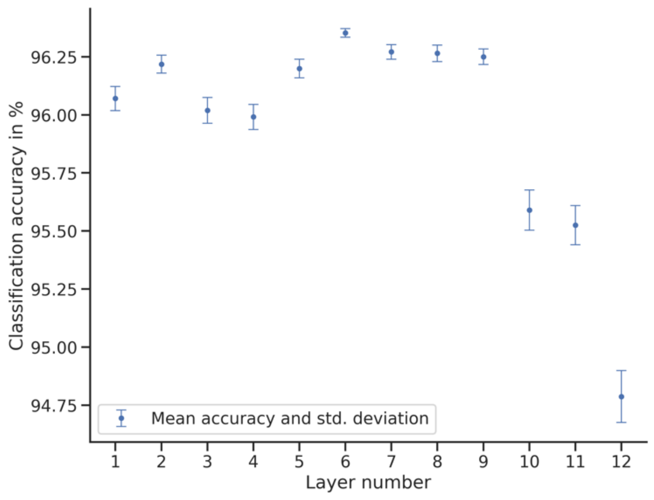

Fig. 5. Results of the experiment for the fixed 50% fault probability.

A. Dataset

The German Traffic Sign Recognition Benchmark (GTSRB) dataset has been in our experiments. It is a dataset for a classification challenge held at the International Joint Conference on Neural Networks (IJCNN) in 2011.

Dataset highlights:

- more than 50 000 images
- 43 classes
- realistic examples with varying lighting conditions and backgrounds

Dataset is split into three subsets:

- Training: 34 799 images
- Validation: 4 410 images
- Testing: 12 630 images

V. RESULTS

Fig. 4 shows results for all experiments. The vertical axis represents the resulted classification accuracy for the fault injection experiments for each of the 12 layers that are placed on the horizontal axis. The varying fault injection probability is color-coded. We can clearly observe that the accuracy of the

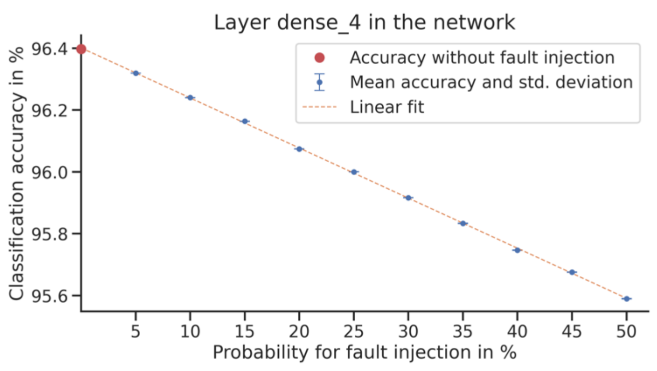

Fig. 6. Results of of the fault injection into the 10th layer.

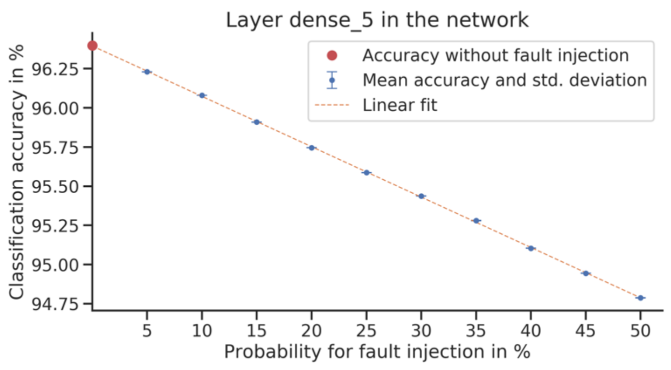

Fig. 7. Results of of the fault injection into the 12th layer.

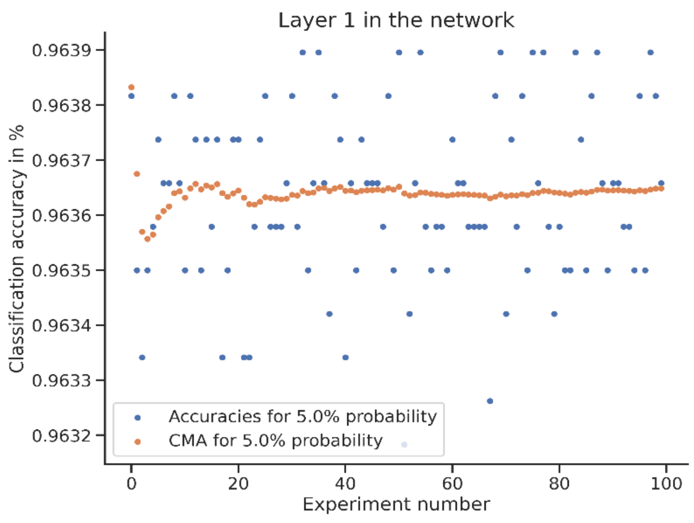

Fig. 8. Thc CMA for the experiments with the first layer.

neural network is proportional to the probability of fault occurrence. The highest decrease in accuracy is observed at the highest 50% probability.

Already in the plot in Fig. 4, we can see that the faults in layers 10, 11, and 12 decrease accuracy more than in the other layers. The results of the experiments for the fixed 50% fault injection probability are shown separately in Fig. 5. Here we can also see the most significant accuracy drop appears on the 10th and 12th layers. These layers are the dense layers. The decrease of the accuracy after the fault injection in these layers are also presented separately in Fig. 6 and Fig. 7.

The drop-out layer 11 also looks critical. However, we have identified that fault injections into a drop-out layer lead to the same effect as fault injections in the previous layer. For example, the fourth drop-out layer and the previous max-pooling layer demonstrate a similar pattern. The reason why this pattern appears is that the drop-out layer does not participate in the validation process of neural network. Drop-out layers are used only during the training process.

We have computed the Cumulative Moving Average (CMA) in order to ensure the statistical confidence of our results. Mathematically, the CMA is an equivalent weighted average of n values. The CMA for the first layer experiments is shown in Fig. 8. We can see that it converges after about 40 experiments. However, 100 experiments have been conducted for each layer.

Our experiments on this CNN have demonstrated the principle feasibility of the InjectTF2. They help to numerically evaluate how random hardware faults such as bitflips drop the overall classification accuracy. The results of these experiments help to identify the most critical layers of the CNN. This information is useful for the intelligent protection of the network. For instance, it would be helpful to triplicate the last three layers of this network like it is shown in Fig. 9. Each of these layers will be executed in parallel, and the results will be compared using the majority voting principle. In this way, e.g., an erroneous output of the layer 11 will not propagate further. Such triplication, of course, will significantly drop the performance of the network. However,

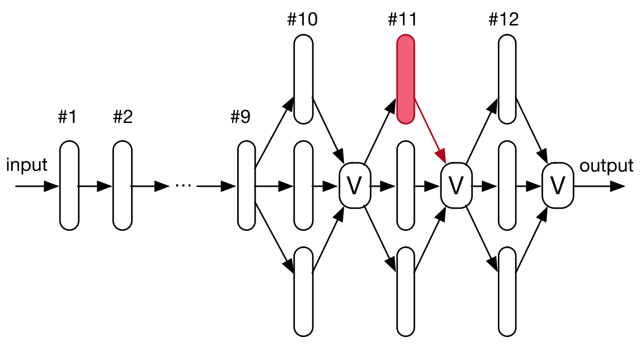

Fig. 9. Recommended triplication of the most critical layers.

the analysis that was demonstrated in this paper will help to identify the most critical layers and protect them selectively.

VI. Conclusion

In this paper, we presented the general description and particular conceptual and implementational features of our recently introduced fault injection framework for TensorFlow 2. In order to demonstrate the application of the framework, we conducted fault injection experiments with a Convolutional Neural Network (CNN). This CNN has been trained using the GTSRB dataset. The fault injection experiments help to identify the most critical layers of a neural network. These results help to introduce smart and selective fault protection mechanisms into the network. Furthermore, the behavior of a network under the influence of faults can be analyzed. Based on the presented results, we proposed a theory, that the reliability of a neural network depends on the overall architecture of the network, the relative position of individual layers, and the parameters (i.e., the number of filters, kernel size, and output dimension) of each layer. In the future, we want to investigate this theory further by comparing more architectures. Also, several improvements of the fault injection frameworks are planned. First of all, we will extend InjectTF2 to support networks with parallel compositions.

REFERENCES

[1] Chen, Chenyi, et al. "Deepdriving: Learning affordance for direct perception in autonomous driving." Proceedings of the IEEE International Conference on Computer Vision. 2015.

[2] Fujiyoshi, Hironobu, Tsubasa Hirakawa, and Takayoshi Yamashita. "Deep learning-based image recognition for autonomous driving." IATSS Research (2019).

[3] Abadi, Martín, et al. "Tensorflow: A system for large-scale machine learning." 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). 2016.

[4] Beyer, Michael, et al. "Quantification of the Impact of Random Hardware Faults on Safety-Critical AI Applications: CNN-Based Traffic Sign Recognition Case Study." *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2019.

[5] Vamsi Ramakrishnan, “Traffic Sign Recognition”, *OnLine: https://github.com/vamsiramakrishnan/TrafficSignRecognition,* Accessed in 2020.

[6] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

[7] Stallkamp, Johannes, et al. "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition." *Neural networks* 32 (2012): 323-332.

[8] Li, Guanpeng, Karthik Pattabiraman, and Nathan DeBardeleben. "Tensorfi: A configurable fault injector for tensorflow applications." *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2018.

[9] TensorFlow Team, “TensorFlow 2.0 is now available!”, *OnLine: https://blog.tensorflow.org/2019/09/tensorflow-20-is-now-available.html*, Accessed in 2019.

[10] Gulli, Antonio and Pal, Sujit. “Deep learning with Keras”, *Packt Publishing Ltd*, 2017.

[11] "YAML Ain't Markup Language (YAML) Version 1.2". YAML.org. Retrieved 2019-05-29.