Bayesian Model for Trustworthiness Analysis of Deep Learning Classifiers

Andrey Morozov[1]* , **Emil Valiev**[2] , **Michael Beyer**[2]
Kai Ding[3] , **Lydia Gauerhof**[4] , **Christoph Schorn**[4]
[1]Institute of Industrial Automation and Software Engineering, University of Stuttgart, Germany
[2]Institute of Automation, Technische Universität Dresden, Germany
[3]Bosch (China) Investment Ltd., Corporate Research, Shanghai, China
[4]Robert Bosch GmbH, Corporate Research, Renningen, Germany
andrey.morozov@ias.uni-stuttgart.de, {emil.valiev, michael.beyer3}@mailbox.tu-dresden.de,
kai.ding@cn.bosch.com, {lydia.gauerhof, christoph.schorn}@de.bosch.com

Abstract

In the near future, Artificial Intelligence methods will inevitably enter safety-critical areas. Deep Learning software, deployed on standard computing hardware, is prone to random hardware faults such as bit flips that can result in silent data corruption. We have performed fault injection experiments on three Convolution Neural Network (CNN) image classifiers, including VGG16 and VGG19. Besides the fact that the bit flips indeed drop the classification accuracy, we have observed that these faults result not in random misclassification but tend to particular erroneous sets of classes. This fact shall be taken into account to design a reliable and safe system. For example, we might consider re-running the classifier if it yields a class for such an erroneous set. This paper discusses the results of our fault injection experiments and introduces a new Bayesian Network (BN) model that aggregates these results and enables numerical evaluation of the performance of the CNNs under the influence of random hardware faults. We demonstrate the application of the developed BN model for the trustworthiness analysis. In particular, we show how to evaluate the misclassification probabilities for each resulting class, for the varying probability of random bit-flips.

1 Introduction

The majority of the high-tech industrial areas already exploit Artificial Intelligence (AI) methods, including deep learning techniques. Presumably, in the next few years, the safety certification challenges of AI components will be overcome, and Deep Learning (DL) will enter safety-critical domains such as transportation, robotics, and healthcare.

*Contact Author. Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

A DL component is simply a piece of software deployed on a standard computing unit. For example, a traffic-sign recognition module of a car receives images from a front camera, detects, and classifies road signs [Beyer *et al.*, 2019]. Such a system is prone to several types of random hardware faults, including bit flips that can occur in RAM or CPU of the computing unit. Bit flips may result in silent data corruption and affect classification accuracy, as shown in [Beyer *et al.*, 2020], [Li *et al.*, 2018]. There are even specific Bit-Flip Attack methods that intentionally cause misclassification by flipping a small number of bits in RAM, where the weights of the network are stored [Rakin *et al.*, 2019] [Liu *et al.*, 2017].

This phenomenon can be investigated with Fault Injection (FI) experiments using methods and tools discussed in Section 2. We have performed such experiments on three Convolution Neural Network (CNN) image classifiers described in Section 3. Besides the fact that the bit flips indeed drop the classification accuracy, we have made another interesting observation: *The injection of a random bit-flip in an output of a particular CNN layer results not in random misclassification, but tends to a particular set of image classes.* For some layers, especially for several first layers, these sets are very distinctive. Examples are shown in Figures 2 and 3. A similar observation was mentioned in [Liu *et al.*, 2017], where the classes from such sets are called the *sink classes*.

This fact has potential practical value and should be taken into account during the reliable and safe design of systems that include DL-components.

- First and the most obvious, if the provided classification result belongs to such a sink set, then we might consider re-running the classifier.
- Second, since these sink sets are different for different CNN layers, we estimate possible fault location, and, for example, re-run the network partially, starting from the potentially faulty layer to reduce computational overhead.
- Third, if several classification results in a row belong to a sink set, then we can assume a "hard" error, e.g., permanent stuck-at one or stuck-at zero in RAM where the data of a particular CNN layer is stored.

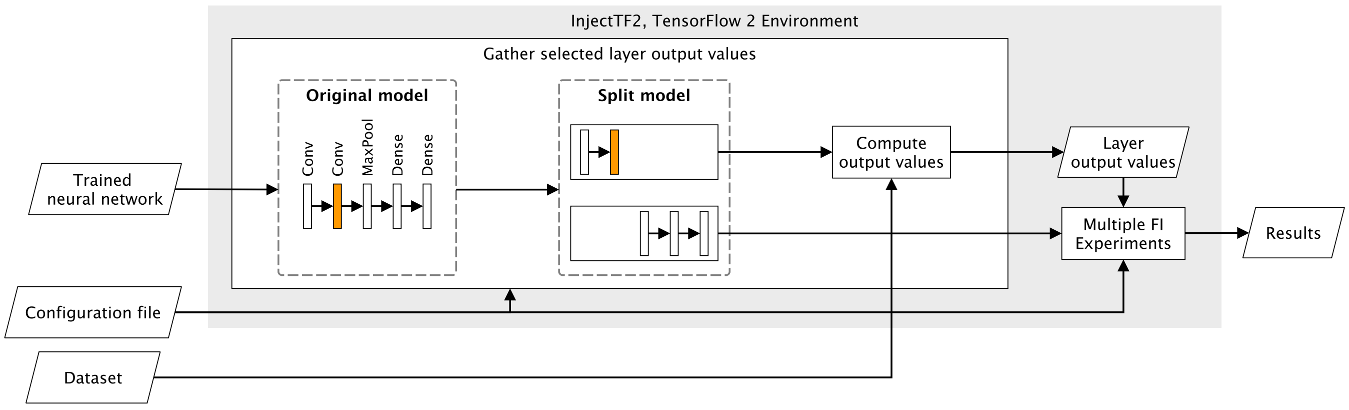


Figure 1: Working principle of the fault injection framework InjectTF2. The layer selected for injection is shown in orange. Source code available at https://github.com/mbsa-tud/InjectTF2.

Contribution: This paper presents the results of the discussed FI experiments. In particular, it shows several examples of sink sets for the layers of VGG16 and VGG19. The complete results of the FI experiments are available online. Based on these experiments, we have developed and fed a Bayesian Network (BN) model that enables numerical evaluation of the performance of the CNNs under the influence of random hardware faults. The paper provides a formal description of this BN model and demonstrates its application for the trustworthiness analysis of the classification results. The paper shows how to evaluate the misclassification probabilities for each resulting class, for the varying probability of random bit-flips.

2 State of the Art

A good overview of the current research effort on making deep learning neural networks safe and trustworthy is given in [Huang *et al.*, 2018]. The authors surveyed the methods for verification, testing, adversarial attack and defense, and interpretability.

In most cases, neural networks are treated as black boxes. Therefore, at the moment, the most straightforward analysis methods are based on fault injection campaigns. The formal verification methods are less found.

Several tools enable automated fault injection into the neural networks. For example, TensorFI [Li *et al.*, 2018] and InjectTF [Beyer *et al.*, 2019] support the first version of TensorFlow. The experiments discussed in this paper were carried out in the TensorFlow V2 environment. Therefore we have used InjectTF2 [Beyer *et al.*, 2020] that was developed for TensorFlow V2. Figure 1 shows the main working principle of the InjectTF2. The tool allows the layer-wise fault injection. InjectTF2 takes a trained neural network, a dataset, and a configuration file as inputs. The network and the dataset should be provided as a HDF5 model and a TensorFlow dataset. In the configuration file, the user can specify the fault type and fault injection probability for each layer of the neural network under test. Currently supported fault types are (i) a random bit flip or (ii) a specified bit flip of a random value of a layer's output.

InjectTF2 performs fault injection experiments in an automated way and logs the results. The model splitting principle sketched in Figure 1, drastically reduces the execution time of the experiments, since the network is not executed from bottom to top each time, but only after the layer where the faults are injected.

In this paper, we are focused on random faults. However, there are also methodologies to evaluate the impact of permanent faults, like the one presented in [Bosio *et al.*, 2019]. Besides that, there are other methods for performance and reliability analysis of deep neural networks that help to improve fault tolerance. A specific fault-injection method for the neural networks deployed in FPGAs and further algorithm-based fault tolerance and selective triplicating of the most critical layers [Libano, 2018]. An efficient bit-flip resilience optimization method for deep neural networks is presented [Schorn *et al.*, 2019].

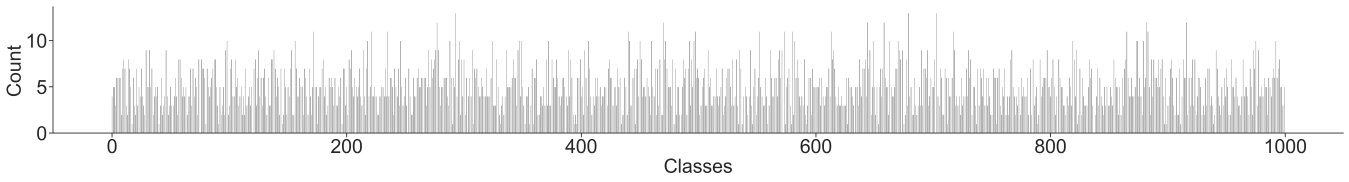
3 Fault Injection Experiments

3.1 CNNs and Datasets

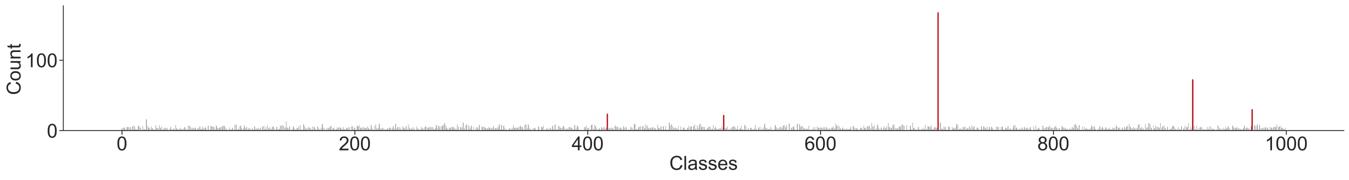
We performed experiments on three different neural networks. The architectures and layer output dimensions of the networks are listed in Table 1.

The first is a self-developed simple CNN, which consists of 12 layers and follows common design principles. The ReLU activation function is used throughout the network, excluding the last layer that uses the Softmax activation function.

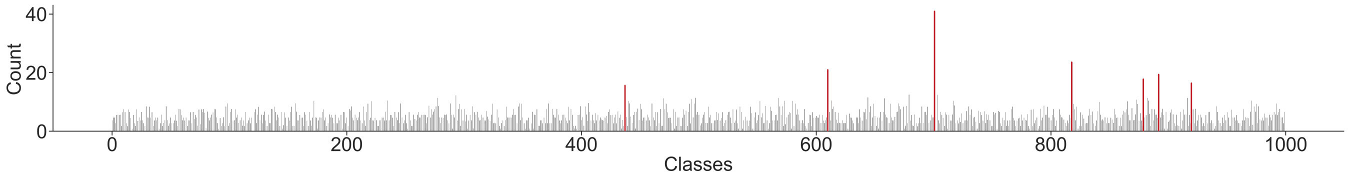
This CNN has been trained on an augmented German Traffic Sign Recognition (GTSRB) [Stallkamp *et al.*, 2012] dataset and can classify road signs with an accuracy of approximately 96 %. The dataset is split into three subsets for training, testing, and validation. The subsets contain 34 799, 12 630, and 4 410 images. Each image has 32×32 RGB pixels and belongs to one of 43 classes of road signs. In order to ensure a uniform classification performance across all classes, the training dataset has been normalized, augmented, and balanced. The augmentation is done by adding copies of images with zoom, rotation, shear, brightness disturbance, and gaussian noise to the dataset. The augmented training subset contains 129 100 images.



(a) VGG16: Fault free run.

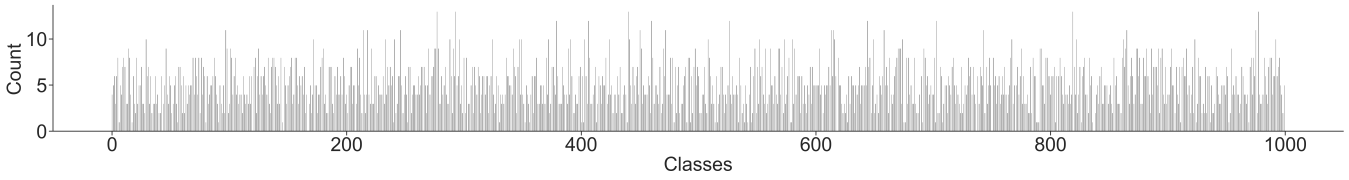


(b) VGG16: Fault injection in Layer 3. The sink classes are highlighted in red.

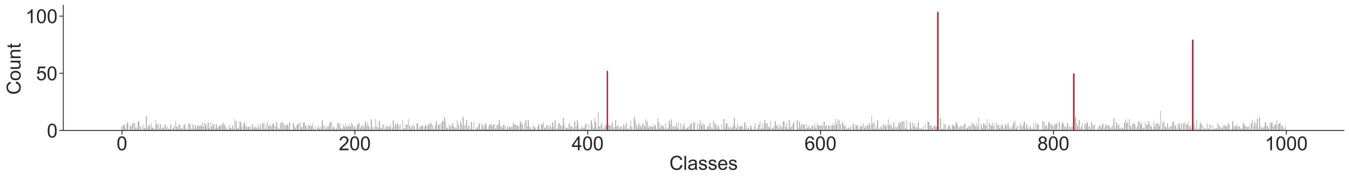


(c) VGG16: Fault injection in Layer 7. The sink classes are highlighted in red.

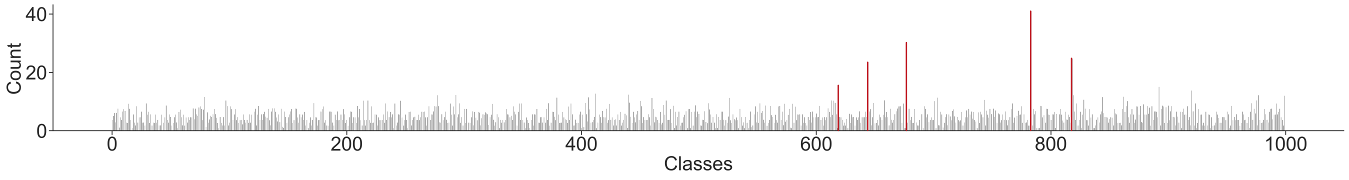
Figure 2: Results of the fault injection experiments on VGG16 for the ImageNet dataset (1000 classes).



(a) VGG19: Fault free run.



(b) VGG19: Fault injection in Layer 3. The sink classes are highlighted in red.



(c) VGG19: Fault injection in Layer 10. The sink classes are highlighted in red.

Figure 3: Results of the fault injection experiments on VGG19 for the ImageNet dataset (1000 classes).

Table 1: The layer structure of the CNNs with output dimensions.

#	**Custom CNN**	**VGG16**	**VGG19**
1	Conv $(32 \times 32 \times 32)$	Conv $(224 \times 224 \times 64)$	Conv $(224 \times 224 \times 64)$
2	Conv $(32 \times 32 \times 32)$	Conv $(224 \times 224 \times 64)$	Conv $(224 \times 224 \times 64)$
3	MaxPool $(16 \times 16 \times 32)$	**MaxPool** $(\mathbf{112} \times \mathbf{112} \times \mathbf{64})$	**MaxPool** $(\mathbf{112} \times \mathbf{112} \times \mathbf{64})$
4	Dropout $(16 \times 16 \times 32)$	Conv $(112 \times 112 \times 128)$	Conv $(112 \times 112 \times 128)$
5	Conv $(16 \times 16 \times 64)$	Conv $(112 \times 112 \times 128)$	Conv $(112 \times 112 \times 128)$
6	Conv $(16 \times 16 \times 64)$	MaxPool $(56 \times 56 \times 128)$	MaxPool $(56 \times 56 \times 128)$
7	MaxPool $(8 \times 8 \times 64)$	**Conv** $(\mathbf{56} \times \mathbf{56} \times \mathbf{256})$	Conv $(56 \times 56 \times 256)$
8	Dropout $(8 \times 8 \times 64)$	Conv $(56 \times 56 \times 256)$	Conv $(56 \times 56 \times 256)$
9	Flatten (4096)	Conv $(56 \times 56 \times 256)$	Conv $(56 \times 56 \times 256)$
10	Dense (256)	MaxPool $(28 \times 28 \times 256)$	**Conv** $(\mathbf{56} \times \mathbf{56} \times \mathbf{256})$
11	Dropout (256)	Conv $(28 \times 28 \times 512)$	MaxPool $(28 \times 28 \times 256)$
12	Dense (43)	Conv $(28 \times 28 \times 512)$	Conv $(28 \times 28 \times 512)$
13		Conv $(28 \times 28 \times 512)$	Conv $(28 \times 28 \times 512)$
14		MaxPool $(14 \times 14 \times 512)$	Conv $(28 \times 28 \times 512)$
15		Conv $(14 \times 14 \times 512)$	Conv $(28 \times 28 \times 512)$
16		Conv $(14 \times 14 \times 512)$	MaxPool $(14 \times 14 \times 512)$
17		Conv $(14 \times 14 \times 512)$	Conv $(14 \times 14 \times 512)$
18		MaxPool $(7 \times 7 \times 512)$	Conv $(14 \times 14 \times 512)$
19		Flatten (25088)	Conv $(14 \times 14 \times 512)$
20		Dense (4096)	Conv $(14 \times 14 \times 512)$
21		Dense (4096)	MaxPool $(7 \times 7 \times 512)$
22		Dense (1000)	Flatten (25088)
23			Dense (4096)
24			Dense (4096)
25			Dense (1000)

The second and third networks are the pre-trained TensorFlow VGG16 and VGG19 [Simonyan and Zisserman, 2014]. They are trained on the ImageNet dataset [Russakovsky *et al.*, 2015]. In the experiments, a random sample of 5000 images from the 2012 ImageNet testing subset has been used. The images belong to 1000 different classes and consist of 224×224 RGB pixels.

3.2 Results

Six examplar bar plots in Figures 2 and 3 describe the classification results for VGG16 and VGG19. The plots display how many images from the input datasets are classified into each of the 1000 classes.

The first (top) plots in both figures show the distributions without faults. The images are distributed more or less uniformly over the classes. The other two plots in each figure show the distributions after the faults injected into specific layers. Precisely, into the layers three and seven of VGG16 and layers three and ten of VGG19. These layers are also highlighted in bold in Table 1. Similar plots for other layers, together with the raw data, are available at https://github.com/mbsa-tud/InjectTF2.

For each layer, we have carried out 100 fault injection experiments. In each experiment, we flip a random bit of a random output value of the corresponding layer. The bar plots represent the average for these 100 experiments.

In the plots, we can observe several distinctive peaks. These peaks reveal that the VGGs tend to erroneously classify images into these sink classes after the fault injections. The peaks are located differently for the presented layers. Note that the peaks are different also for the third layers of VGG16 and VGG19.

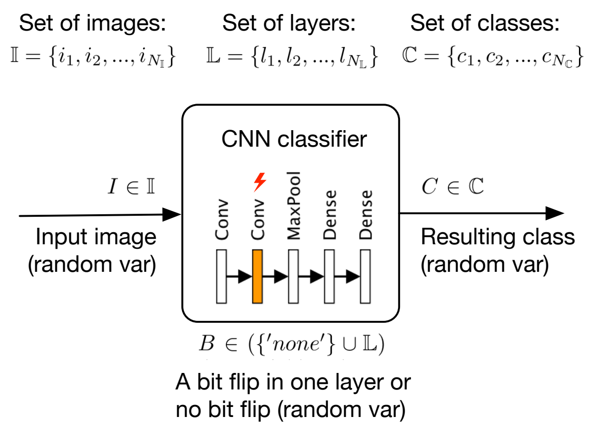
However, for several layers, especially from the same VGG blocks, the peaks are quite similar. We also observed that such peaks appear only in the first layers, and the misclassification became more random if we inject faults into the more in-depth layers. Note that the peaks for the seventh and tenth layers are lower than the peaks of the third layer. The peaks are distinctive for the first 11 layers of VGG 16 and the first 12 layers of VGG19. After that, the distribution becomes more or less uniform.

4 Trustworthiness analysis

The experimental results discussed above enable numerical evaluation of the performance of the CNNs under the influence of random hardware faults. For instance, we can statistically evaluate the probability of misclassification for each resulting image class. This probability is higher for the sink classes than for other classes. For this purpose, we use a Bayesian Network (BN) model fed with the statistical results of the fault injection experiments.

4.1 Formal Model of the Classifier

The BN is defined using a formal set-based model of a classifier that is shown in Figure 4. This model is based on three sets, two functions, and three random variables.



Set of images:
$\mathbb{I} = \{i_1, i_2, ..., i_{N_\mathbb{I}}\}$
Set of layers:
$\mathbb{L} = \{l_1, l_2, ..., l_{N_\mathbb{L}}\}$
Set of classes:
$\mathbb{C} = \{c_1, c_2, ..., c_{N_\mathbb{C}}\}$
CNN classifier
$I \in \mathbb{I}$
Input image (random var)
Conv
Conv
MaxPool
Dense
Dense
$C \in \mathbb{C}$
Resulting class (random var)
$B \in (\{'none'\} \cup \mathbb{L})$
A bit flip in one layer or no bit flip (random var)

Figure 4: Formal set-based model of a classifier.

Sets:

$\mathbb{I} = \{i_1, i_2, ..., i_{N_\mathbb{I}}\}$ - Input images.
$\mathbb{C} = \{c_1, c_2, ..., c_{N_\mathbb{C}}\}$ - Result classes.
$\mathbb{L} = \{l_1, l_2, ..., l_{N_\mathbb{L}}\}$ - Layers of the CNN.

Functions:

$g: \mathbb{I} \times \mathbb{C} \rightarrow \{1,0\}$ - Formalization of the results of the fault-free run, $g(i,c) = 1$ if image i is classified as class c, and $g(i,c) = 0$ otherwise. For simplicity, we assume that classification is always correct.

$f_k^l: \mathbb{I} \times \mathbb{C} \rightarrow \{1,0\}$ - Formalization of the results of the FI experiments, $f_k^l(i,c) = 1$ if image i is classified as class c in the k^{th} FI experiment, the faults are injected in layer $l \in \mathbb{L}$, $f_k^l(i,c) = 0$ otherwise.

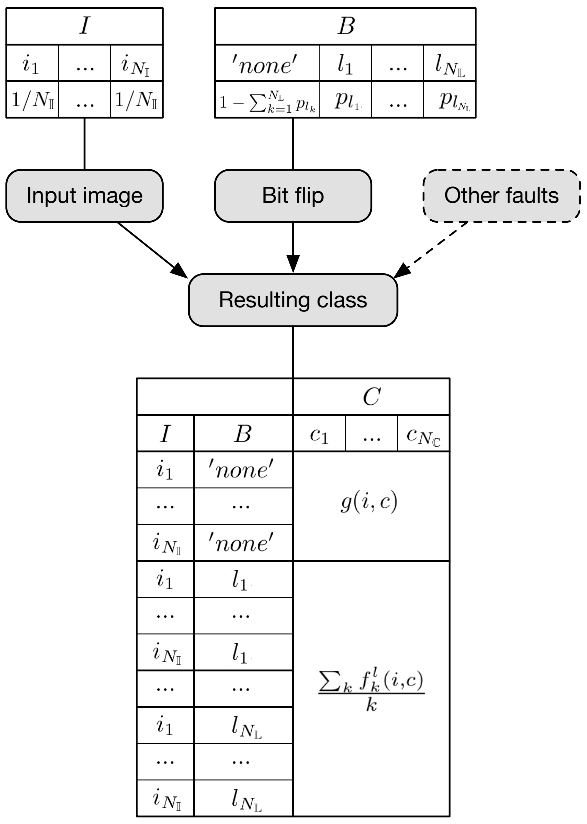
Random variables:

$I \in \mathbb{I}$ - Current input image. An independent discrete random variable. For simplicity, we assume that the probability that an image from $\mathbb{I}$ is the input image is equal for all images: $P(I = i) = 1/N_\mathbb{I}, \forall i \in \mathbb{I}$. Otherwise the distribution should be specified statistically.

$B \in \{'none'\} \cup \mathbb{L}$ - No bit flip or a bit flip in a particular layer. An independent discrete random variable. The value $'none'$ means that there was no bit flip during the run. A value $l \in \mathbb{L}$ means that it was a bit flip in layer l. We assume, that only a single bit flip can happen during the run. The distribution is defined according to a variable p_{l_k} that defines a probability of a bit flip in a layer l_k. For the simplicity we will apply the same probability p for each layer. The outcome $'none'$ is defined as the complement of all other events: $P(B =' none') = 1 - \sum_{k=1}^{N_\mathbb{L}} p_{l_k}$.

$C \in \mathbb{C}$ - Resulting class. A discrete random variable that depends on I and B.

4.2 Bayesian Network



I		
i_1	...	$i_{N_\mathbb{I}}$
$1/N_\mathbb{I}$	...	$1/N_\mathbb{I}$

B			
$'none'$	l_1	...	$l_{N_\mathbb{L}}$
$1 - \sum_{k=1}^{N_\mathbb{L}} p_{l_k}$	p_{l_1}	...	$p_{l_{N_\mathbb{L}}}$

		C		
I	B	c_1	...	$c_{N_\mathbb{C}}$
i_1	$'none'$	$g(i,c)$		
...	...			
$i_{N_\mathbb{I}}$	$'none'$			
i_1	l_1	$\frac{\sum_k f_k^l(i,c)}{k}$		
...	...			
$i_{N_\mathbb{I}}$	l_1			
...	...			
i_1	$l_{N_\mathbb{L}}$			
...	...			
$i_{N_\mathbb{I}}$	$l_{N_\mathbb{L}}$			

Figure 5: The Bayesian network and conditional probability tables.

A BN is a graphical formalism for representing joint probability distributions [Pearl, 1985]. It is a probabilistic directed acyclic graph that represents a set of variables and their conditional dependencies. Each node is defined with a Conditional Probability Table (CPT). Our BN describes the conditional probabilities of C. Figure 5 shows the BN and the CPTs of the three random variables. The CPTs of independent variables I and B define constant probabilities for each outcome. The outcome of C depends on the outcomes of I and B. So the probabilities are defined for each combination of the outcomes of I and B.

The CPT of C is divided into two parts. The upper part describes the situation without bit flips. We assumed perfect classification. Thus, this part consists just of zeroes and ones. The ones indicate the correct classes for each image. Mathematically we represent this using the function g: $p_{i,'none',c} = g(i,c)$. The bottom part describes the situation when bit flips occur in corresponding layers. Here we statistically approximate the probabilities using the results of our fault injection experiments. Mathematically we represent this using the function f. Each probability is estimated as the number of times when i was classified as c divided by the total number of the fault injection experiments ofr the layer l: $p_{i,l,c} = \sum_k f_k^l(i,c)/k$.

4.3 Quantification

The BN stores the results of the fault injection experiments in a structured way. This allows the analysis of various reliability-related properties. From the general cumulative probability of misclassification to the specific probabilities of the wrong classification of a particular input image because of a bit flip in a particular layer. Moreover, other kinds of random faults and their combinations can be taken into account, as shown in Figure 5 with the dashed lines.

As an example, we show how to quantify the trustworthiness for each resulting class. We define the trustworthiness as a kind of inverse probability, the probability that the resulting class c is the correct class for the input image i, taking into account the possibility of a bit flip in any layer. $\mathbb{I}_c$ is a subset of images that belong to the class c: $\mathbb{I}_c \subset \mathbb{I} : i \in \mathbb{I}_c$ if $f(i, c) == 1, i \in \mathbb{I}, c \in \mathbb{C}$. Then, the trustworthiness of class c is the conditional probability $P(I \in \mathbb{I}_c \cap B \in \mathbb{B}|C = c)$. Applying first the formula for conditional probability (Kolmogorov definition) and then the law of total probability, we obtain the following expression.

$$P(I \in \mathbb{I}_c \cap B \in \mathbb{B}|C = c) =$$
$$= \frac{P(C = c \cap I \in \mathbb{I}_c \cap B \in \mathbb{B})}{P(C = c)} =$$
$$= \frac{\sum\limits_{i \in \mathbb{I}_c} \sum\limits_{b \in \mathbb{B}} P(C = c|B = b \cap I = i)P(B = b)P(I = i)}{\sum\limits_{i' \in \mathbb{I}} \sum\limits_{b' \in \mathbb{B}} P(C = c|B = b' \cap I = i')P(B = b')P(I = i')}$$

Where, $P(I = i)$ is from the CPT of I, $P(B = b)$ is from the CPT of B, and $P(C = c|B = b \cap I = i)$ is from the CPT of C. In the numerator of the fraction, we sum up only for i from $\mathbb{I}_c$ and for all i from $\mathbb{I}$ in the denominator. Basically, we compute the ratio of correct classifications to all classification.

In our experiments, we computed the probabilities with our self-developed scripts. However, probabilistic analytical software libraries, like pomegranate [Schreiber, 2018], allow efficient and scalable methods for computation of Bayesian networks.

4.4 Results

Figures 6, 7, and 8 show misclassifcation probabilities for each class. These probabilities are computed as one minus the trustworthiness. The probabilities of bit flips vary from 10^{-7} to 1 for the custom CNN and from 10^{-11} to 10^{-4} for the VGG16 and VGG19. Ten classes with the highest misclassification probabilities are highlighted with colors (sorted using the results obtained for the probabilities of bit flip 10^{-9}). The remaining classes are shown in grey.

Based on the estimated bit flip probabilities, we can decide whether we trust the classification result or not. Moreover, from the safety point of view, the misclassification for some classes might be more hazardous than for the others. For instance, it might be more critical to confuse the stop sign with the main road sign than to confuse speed limits 30 and 50. Such cases can be easily quantified using the proposed Bayesian model. They could also lead to re-training regarding the classes with the lowest trustworthiness.

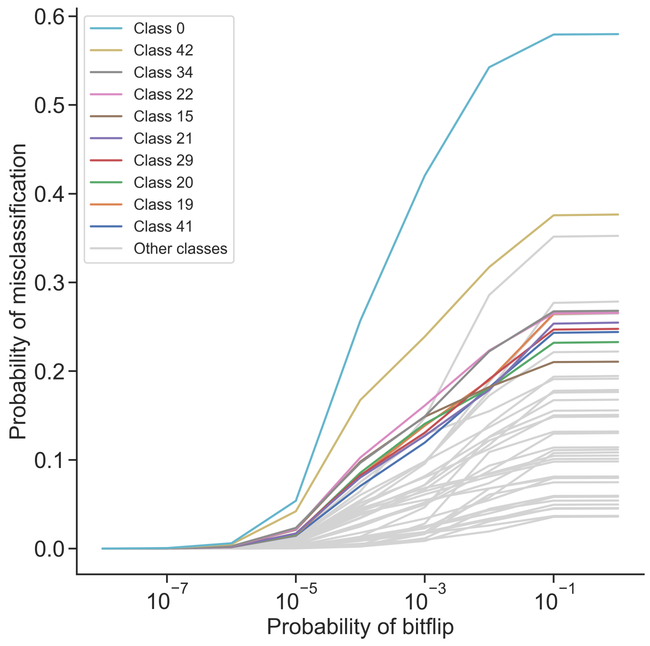


Figure 6: Misclassification probabilities for the varying probability of a bit flip for the image classes of the custom CNN.

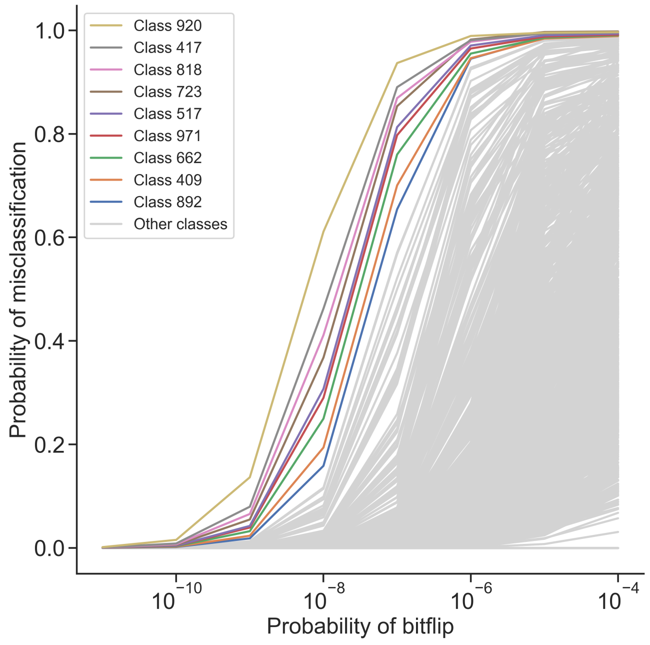


Figure 7: Misclassification probabilities for the varying probability of a bit flip for the image classes of the VGG16.

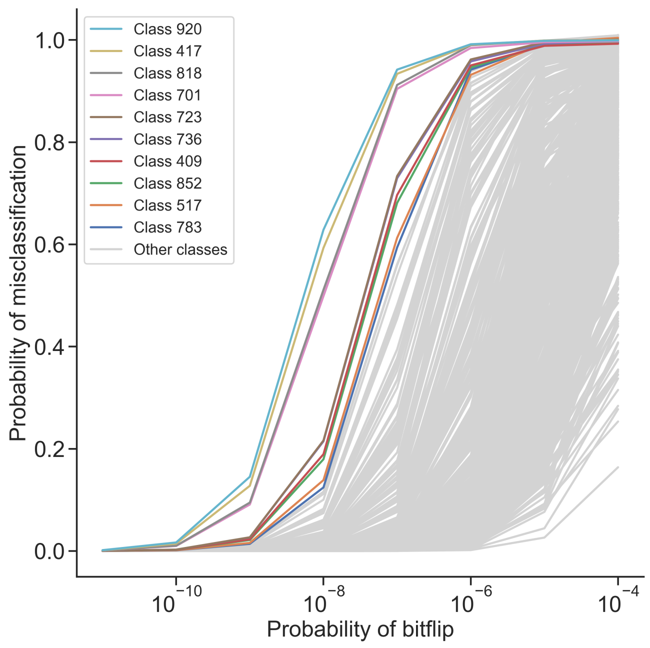


Figure 8: Misclassification probabilities for the varying probability of a bit flip for the image classes of the VGG19.

5 Conclusion

A series of fault injection experiments on several CNN-based classifiers have shown that random hardware faults result not in random misclassification but tend to misclassify the input images into specific distinctive sets of classes. These sets are different for functionally equivalent CNNs. Also, these sets depend on the layer where a fault is injected. This information has to be taken into account during the reliability and safety analysis of such classifiers if they shall be integrated into a safety-critical system. In this paper, we proposed the application of a Bayesian network model fed with the results of such fault injection experiments. This model allows a broad range of numerical reliability and safety-related analysis of the classifier under test. As an application example, we have demonstrated how the proposed Bayesian model helps to estimate the level of trustworthiness for each resulting image class.

References

[Beyer *et al.*, 2019] M. Beyer, A. Morozov, K. Ding, S. Ding, and K. Janschek. Quantification of the impact of random hardware faults on safety-critical ai applications: Cnn-based traffic sign recognition case study. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 118–119, Oct 2019.

[Beyer *et al.*, 2020] Michael Beyer, Andrey Morozov, Emil Valiev, Christoph Schorn, Lydia Gauerhof, Kai Ding, and Klaus Janschek. Two fault injectors for tensorflow: Evaluation of the impact of random hardware faults on vggs. In *The paper is submitted to EDCC2020. Under evaluation. This will be updated in the final version of the paper.*, pages xxx–xxx, XXX, XXX, XXX 2020. XXX.

[Bosio *et al.*, 2019] A. Bosio, P. Bernardi, A. Ruospo, and E. Sanchez. A reliability analysis of a deep neural network. In *2019 IEEE Latin American Test Symposium (LATS)*, pages 1–6, 2019.

[Huang *et al.*, 2018] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks. *arXiv preprint arXiv:1812.08342*, 2018.

[Li *et al.*, 2018] Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. Tensorfi: A configurable fault injector for tensorflow applications. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 313–320. IEEE, 2018.

[Libano, 2018] Fabiano Pereira Libano. Reliability analysis of neural networks in fpgas, 2018.

[Liu *et al.*, 2017] Y. Liu, L. Wei, B. Luo, and Q. Xu. Fault injection attack on deep neural network. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 131–138, 2017.

[Pearl, 1985] Judea Pearl. Bayesian netwcrks: A model cf self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine, CA, USA*, pages 15–17, 1985.

[Rakin *et al.*, 2019] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1211–1220, 2019.

[Russakovsky *et al.*, 2015] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[Schorn *et al.*, 2019] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. An efficient bit-flip resilience optimization method for deep neural networks. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1507–1512. IEEE, 2019.

[Schreiber, 2018] Jacob Schreiber. Pomegranate: fast and flexible probabilistic modeling in python. *Journal of Machine Learning Research*, 18(164):1–6, 2018.

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

[Stallkamp *et al.*, 2012] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012.