POLTECNICO DI TORINO

Master's Degree in Data Science and Engeneering

Politecnico
di Torino

Master's Degree Thesis

Reliability evaluation of Split Computing Neural Networks

Supervisors

Matteo SONZA REORDA

Juan David GUERRERO BALAGUERA

Josie Esteban RODRIGUEZ CONDIA

Candidate

Giuseppe ESPOSITO

December 2023

Abstract

In the contemporary era, Artificial Intelligence (AI) has become integral to Internet of Things (IoT) systems, revolutionizing several fields. Due to resource constraints of these devices, various model optimization techniques are employed, such as split computing (SC), where the workload is partly offloaded to the cloud to ensure that the required resources are within the capabilities of the employed devices. Despite these optimizations, models still require advanced hardware like GPUs which may be affected by faults. The Graphics Processing Unit, or GPU, has emerged as a vital computing technology in both personal and business settings. It is specifically designed for parallel processing and is utilized in numerous applications such as graphics and video rendering. Nevertheless, GPUs are becoming more popular for use in creative production and artificial intelligence (AI), due to their capability to speed up computation, in case it involves simpler basic operations. The employment of such advanced hardware brings itself some risks particularly when they are used for safety-critical application. To assess the reliability of this sensitive hardware, Fault Injection simulations are carried out in order to find a relationship between the performance of the model and the features of its corruption. The evaluation of SC2 models reliability in the presence of GPU faults remains unexplained. This thesis work examines the impact of hardware failures on system reliability under the assumptions of the split computing approach that distributes neural network architecture between mobile devices and cloud systems in addition to a knowledge distillation process to maintain prediction accuracy and reduce transmission load that could be degraded by the injection of an artificial bottleneck. Furthermore, the research identifies vulnerable features through software-level simulations and investigates various hardening techniques on lightweight Deep Neural Network models. The findings exhibit substantial deterioration of accuracy in models subjected to fault injection during inference of several tasks, including image classification, object detection, and semantic segmentation. Suggested hardening techniques, including custom activation functions, show promise in improving model robustness and several simulation campaigns have been carried out and the corresponding statistics analyzed and compared.

Acknowledgements

I would like to thank my supervisors Matteo and tutors Juan David and Josie Esteban for their constant guidance, invaluable insights and continuous support during the research project. Their expertise, constructive feedback and commitment have been instrumental in shaping this project and enhancing my understanding of the subject matter. I would like to thank my family who have been a guiding light in the night when everything seemed to be hard. Your moral support guided me to my goals and for that I can only be grateful. A big thank you to my friends, whose inspiration and motivation have made the challenging moments of this academic endeavour joyful and bearable. Your laughter and support have made this journey memorable and rewarding.

Giuseppe.

Computing resources provided by hpc@polito (http://www.hpc.polito.it).

Table of Contents

List of Tables

List of Figures

Acronyms

AI
artificial Intelligence

NN
Neural Network

COTS
Commercial-Off-The-Shell

ANN
Artificial Neural Network

CNN
Convolutional Neural Network

DNN
Deep Neural Network

FC layer
Fully-connected layer

ReLU
Rectified Linear Unit

ILSVRC
ImageNet Large Scale Visual Recognition Challenge

EC
Edge Computing

EE

Early Exiting

MLE

Maximum Likelihood Estimation

FLOPS

Floating Point Operations Per Second.

FI

Fault injection

SUT

System under testing

IoU

Intersection over Union

FAT

Fault Aware Training

HDL

Hardware Description Level

IoT

Internet of Things

LSB

Lowest Significant Bit

MSB

Most Significant Bit

GeMM

General Matrix Multiplication

SC

Split Computing

SC2

Supervised Compression for Split Computing

SDC

Silent Data Corrupted

MSE

Mean Squared Error

mAP

mean Average Precision

SGD

Stochastic Gradient Descent

RMSProp

Root Mean Squared Propagation

RCNN

Region-based Convolutional Neural Network

FPN

Feature Pyramid Network

SSD

Single Shot multiblox Detector

GPU

Graphics Processing Unit

VGG

Visual Geometry Group

MM

Matrix Multiplication

SM

Streaming Multiprocessor

FMA

Fused Multiply–Add

FCNN

Fully Convolutional Neural Network

SPP

Spatial Pyramid Pooling

ASPP

Atrous Spatial Pyramid Pooling

VOC

Visual Object Classes

HardTanH

Hard Hyperbolic Tangent

CR+BQ

Channel Reduction + Bit Quantization

MRAD

Mean Relative Accuracy Degradation

Chapter 1

Introduction

1.1 Motivation

In the present era, Artificial Intelligence (AI), in particular, Deep Neural Networks (DNNs), has become an integral element within Internet of Things (IoT) systems, enhancing their capacity to improve new technologies in the fields of public services, industry, automotive, and healthcare (among the others). Consider, for instance, the utilization of object detection models in self-driving cars to identify obstacles, or the application of trajectory tracking models in the same scenario, aiding in following road markings.

Aside from the substantial computational resources required for training employed DNN models for these tasks, the computational cost incurred solely during the inference stage must not be underestimated. For instance, resource-intensive models typically demand the utilization of high-performance computing systems or powerful Graphic Processing Units (GPUs) devices. However, compact devices such as IoT systems are intended to have limited computational capabilities, since they typically correspond to Commercial Off-The-Shell (COTS). Hence, such that running DNN algorithms becomes unfeasible with the Edge Computing. Consequently, the idea of completely offloading the computational cost to cloud systems has been taken into account.

Nonetheless, state-of-the-art techniques see the advantages of Split Computing techniques. It allows to split the model's architecture in *Head* which is executed on the mobile device (i.e., IoT device), and *Tail* executed on the cloud or on an Edge server. Furthermore, in order to minimize the transfer load to the cloud as much as possible, a layer or a block of bottleneck layers replace a layer in the *split point*, whose aim is to significantly reduce the number of neurons as well as to reduce the amount of data interchange between the mobile device and the cloud.

As the tail architecture receives a compressed version of the feature maps, a

knowledge distillation process is performed in order to train a compressed version of the model, i.e., the head. The knowledge distillation process aims to replicate the knowledge that the model would have had if the bottleneck had not been introduced. This involves training a compressed version of the original model, making it lighter while maintaining the same performance as far as possible.

While these advancements will keep playing a subordinate role in human decision-making, some dangerous situations can arise when hardware malfunctions come into play, if we consider the aforementioned case, hardware faults can lead to the wrong detection of an object close to the autonomous vehicle or, even worse, an accident caused by the missing detection of a person. Thus, assessing the dependability of these technologies, particularly their resilience against such errors, becomes of paramount importance, in order to enhance the fault resilience of such systems when using the Split Computing paradigm.

The objective of this work is first to develop a Fault Injection system for reliability and robustness assessment to faults that could occur at the hardware level. The fairness tests of such Fault Injector are enforced by expanding the selection of models already available in Supervised Compression for Split Computing (SC2) with the possibility of being adapted to new datasets. As soon as a general assessment of how the different characteristics of the faults can affect the performance of the software, some software hardening techniques are then proposed based on the previous analysis, together with a deep understanding of the structure of the models under testing.

1.2 Problem formulation

A flexible framework for training and evaluating models in the split computing paradigm was proposed by Yoshitomo Matsubara et al. in [1]. Their contribution to the state of the art was in the development of a new training technique based on Supervised Compression, which focuses on reducing the size of the model on the mobile device side by training it to mimic the behavior of an already trained and heavier version of the same.

This thesis work presents a framework to perform application-level fault injection campaigns to induce error injection at both the weight and neuron levels of any split computing DNN, allowing for a considerable degree of flexibility. Furthermore, we created additional SC DNN architectures using different configurations and evaluation datasets that allow us to explore the incidence of hardware faults on their reliability.

A significant gap in the current landscape is the lack of a benchmark for assessing the resilience of models within the split computing framework. As this framework is proposed for use on IoT devices, it is a pressing issue that necessitates the

attention of the research community. Developing a uniform benchmark will not only permit comparative analyses but also play a vital role in guaranteeing the practicability and dependability of fault features with split computing models in real-world situations.

1.3 Thesis contribution

My contribution to this research work is:

- Analyzing the results of software-level simulation of fault injection and provide a comprehensive overview of the most vulnerable software features, in order to enhance their protection and hardening.
- Extending the selection of models trained with Supervised Compression for Split Computing (SC2) by adapting existing architectures to the split computing paradigm and training them on new datasets.
- Exploring, implementing and testing different hardening techniques in order to improve the robustness of lightweight Deep Neural Network models for image classification, object detection and semantic segmentation tasks.

This thesis is divided as follows:

- *Chapter 1*: Provides an overview of the reasons behind the project and the main contributions of this specific work.
- *Chapter 2*: Introduces the basis of both machine learning algorithm and fault injection background, clarifying the tools applied during the development.
- *Chapter 3*: Explores different fault injection techniques at the hardware level in GPU accelerators and software level in deep learning applications.
- *Chapter 4*: Detailed description of the experimental setup for the fault injection, applied methodologies for the adaptation of new models to the SC2 framework, and software reinforcing techniques implementation.
- *Chapter 5*: Results of the analysis aimed at finding a correlation between the hardware fault features and the trend of the metrics
- *Chapter 6*: Conclusions of the thesis work.

Chapter 2

Background

2.1 Machine learning fundamentals

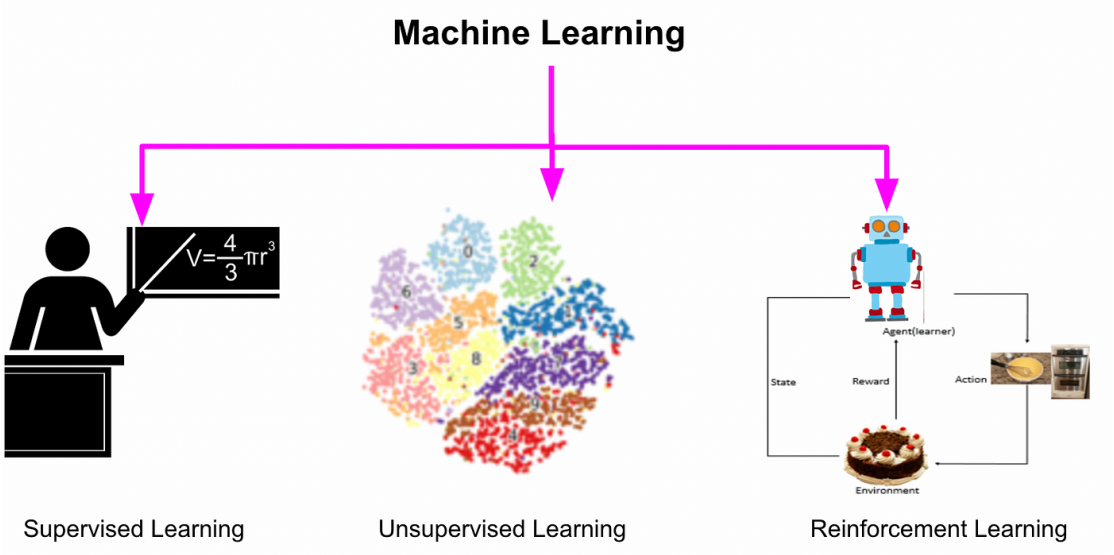


Figure 2.1: Main subdivision of machine learning algorithms, [Source]

Machine Learning is a branch of Artificial Intelligence that aims to mimic a human-like learning process. Human reasoning is composed of both deductive and inductive reasoning: *deductive reasoning* entails applying general rules or principles to arrive at specific conclusions. For example, if we know that "all men are mortal" and that "Socrates is a man," we can deduce that "Socrates is mortal.". On the other hand, *inductive reasoning* is based on the generalization of patterns discovered in situations already experienced. However, *Inductive Reasoning* extracts

useful relations between data and attributes by means of statistical models. For example, over the past thousands of years, scientists have observed that water freezes at 0 Celsius degrees at standard atmospheric pressure. Considering that the available observations come from various geographic locations and climates which consistently support the first premise, it can be inferred that under the above-mentioned conditions, regardless of the geographic location, the water freezes at or below 0 degrees Celsius. In the same way that these forms of reasoning lead humans to conclusions based on empirical evidence, it is crucial to highlight the parallelism between *inductive reasoning* and the *learning process* of a machine learning model.

Machine learning can be divided into 3 main categories [2]:

- **Supervised learning**. Algorithms for supervised learning use a dataset comprising features with an additional dimension where a target or label for each occurrence is reported. As an example, consider the Iris dataset[1], where each Iris plant is labeled with the appropriate species. This dataset may be examined by a supervised learning algorithm, which can then learn to classify Iris plants into three different species based on the available *ground truth*.

- **Unsupervised Learning**. Unsupervised learning algorithms interact with a dataset containing a variety of features and eventually gain useful information about the dataset's underlying structure. In the field of deep learning, our typical goal is to capture the probability distribution that produced the dataset. For activities like anomaly detection or denoising, such algorithms can be employed in order to perform *pattern recognition* and group the elements on the dataset based on possible hidden trends.

- **Reinforcement learning**. Algorithms for Reinforcement learning are not trained from the experience of a fixed dataset, they interact with the environment. Specifically, Fig 2.1 an *Agent A* interacts with an *Environment E* with the default setup and then a *Quality Function* is designed such that A is able to accomplish the assigned learning task.

2.1.1 Deep Learning

Deep Learning is a subset of Machine learning that includes special algorithms: **Artificial Neural Networks** (ANNs). The concept of ANN dates back to the 1940s and 1950s when they were inspired by models of biological Neural Networks (NNs) in the brain. Nowadays, NNs are software-implementable algorithms and

[1]A tabular dataset which contains information about sepal length, petal length, sepal width and petal width of three categories of Iris dataset.

they are very flexible. It means that, depending on the *architecture* of the algorithm, they can perform different tasks.

All the possible tasks can be grouped into several macro areas, but the ones that most influence our everyday lives are: Computer Vision, Natural Language Processing, and Time Series Forecasting. This section will focus on Computer Vision and specifically on the tasks that include the models that have been part of the objective of the following study.

Computer vision plays a major role in today's technological landscape. It aims to replicate the advanced human capability for visual perception by giving machines the ability to perceive the visual world as humans do. Possible real-world applications of such a technology are autonomous vehicles, medical imaging, security and surveillance, industrial product inspection, navigation and mapping, art and creativity, and agriculture. Each goal can be reached by changing the internal structure of the NN; nevertheless, the most frequent layer types are the Convolutional Layers and the Fully-Connected layers.

Let us now examine a toy example in order to understand the general idea of Convolutional Neural Networks(CNNs). Given two any functions $f(x)$ and $g(x)$ the **Convolution** can be expressed as the sum of the overlaps of one of the functions and all of the shifted versions of the other function:

$$(f \times g)(x) = \int_{-\infty}^{\infty} f(t) \times g(x-t)dt \tag{2.1}$$

Commonly structured CNN receives a d-dimensional tensor[2] as input, which represents the image to be processed. The first stages of the algorithm process form the feature extractor and are typically performed by Convolutional layers. Specifically, a dot product[3] between the tensor and some generally small matrices called **kernels** is computed, which represents the overlap of the two functions described in Eq. 2.1, followed by an activation function and possibly pooling operation. Eventually, d-dimensional tensors are extracted and are called **feature maps**.

An *activation function* is a mathematical function that adds non-linearity to the objective function to optimize, which enables the model to simulate a non-linear mapping input-output by deciding whether to fire the response of a neuron or not and assigning a corresponding weight. On the other hand, the feature map's spatial dimensions are lowered during a *pooling* operation by choosing the most important data from smaller locations, preserving key features while lowering computing complexity. This sequence is usually known as a **Convolutional Block**. The extracted feature maps typically represent edges, textures, or shapes which become more detailed the more convolutions are performed. This means the higher the

[2]A multidimensional array.

[3]Given 2 vectors $\underline{a}$ and $\underline{b}$ the *dot product* $(\cdot)$ is defined as $\underline{a} \cdot \underline{b} = a_1b_1 + a_2b_2 + \ldots + a_nb_n$

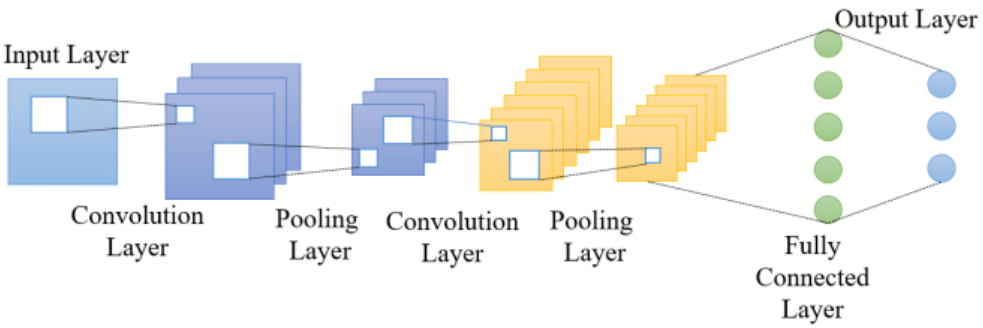


Figure 2.2: Basic architecture of a general-purpose Convolutional Neural Network

feature resolution, the better the classification capabilities of the model on the object under analysis. Nevertheless, the more convolutions applied, the more the NN loses the perception of the spatial location of the object depicted in the image.

The objective of the whole process is to encode images as feature maps, which makes it simpler for the NN to identify patterns and objects. Usually, the concatenation of Convolutional Blocks represents the backbone of the model and the process is called *feature extraction*. Subsequently, they are concatenated with some other type of layers: *Fully-Connected layers*. Fully-Connected layers perform the dot product of the feature maps with a vector of weights ($\underline{w}$,$\underline{b}$):

$$y = f\left(\underline{w} \cdot x + \underline{b}\right)$$

The last convolutional layer output is then passed through a dense layer block ending with a *softmax activation function* which outputs a probability distribution. Such a distribution quantifies the confidence level associated with each possible label, into the input image. Overall, the depth of the NN depends on the number of layers that are inside its architecture.

As the aim of this work is to create a baseline for the field of interest, we decided to focus on the 3 most frequent tasks of Computer Vision.

- **Image Classification**: it is a Supervised task of Computer Vision where the goal is to assign a label to an input image. By definition, the dataset of images is provided with target labels to assign to each image. The toy example explained above is the typical pipeline of inference of a model meant for image classification. Indeed, one of the main obstacles involves the need for extensive and accurately annotated datasets to facilitate training. The processes of collecting, labeling, and maintaining these expansive datasets can consume significant time and resources. Additionally, reaching elevated

accuracy levels calls for intricate models, often demanding substantial computational capabilities. This situation concerns energy consumption and the potential for scalable implementation.

- **Object Detection**: it is a pioneering field in computer vision that has revolutionized the way machines perceive and understand visual data. Unlike image classification which assigns a single label to an entire image, object detection goes a step further by identifying and localizing multiple objects within an image, providing valuable information about their positions and dimensions, [3]. The importance of object detection is evident in its wide range of applications. In autonomous driving, it enables vehicles to identify pedestrians, other vehicles, and traffic signs, enhancing road safety. In healthcare, it assists in detecting anomalies in medical images, such as tumors in X-rays. Retail and e-commerce use object detection for inventory management and visual search capabilities [4].

- **Semantic Segmentation**: it involves the partitioning of an image into distinct segments, each corresponding to a specific object or region of interest. Unlike object detection which identifies objects as bounding boxes, semantic segmentation assigns a label to every pixel in an image, providing a detailed understanding of the image's content. The significance of semantic segmentation lies in its ability to enable machines to comprehend images at a more granular level [5]. This has different applications, ranging from medical image analysis, where it aids in pinpointing specific structures like tumors, to environmental monitoring, where it assists in land cover classification through satellite imagery. In autonomous driving, semantic segmentation helps vehicles understand road scenes by identifying different objects and road markings [6] [7].

The *backbone* of the models employed for computer vision tasks can have different architectures depending on the support that these features can provide to the downstream task. Over the past ten years, numerous models have been developed and are now utilized as benchmarks for feature extraction. The following subsection will focus on the models that have recently influenced the advancement of new backbones and specifically the advantages that such innovation has brought.

2.1.2 VGG

VGG feature extractor is the result of an experiment carried out by Karen Simonyan and Andrew Zisserman who wanted to prove that by pushing the depth of a feature extractor to 16-19 layers, the classification capabilities of the model improve a lot with respect to the state-of-the-art. In [8] they showed the architecture that

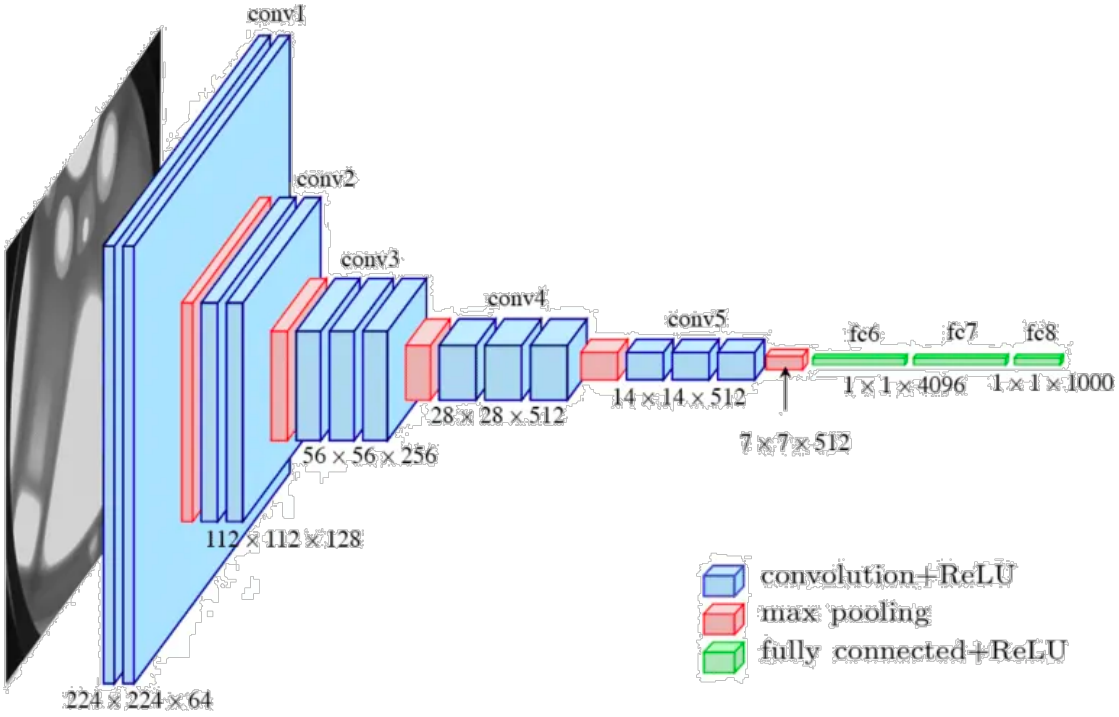


Figure 2.3: VGG 19 backbone plus several ending dense layers for the classification. [Source].

is reported in Figure 2.3. The small sizes of the kernels with the sharp increase of the number of channels of the feature maps preceded by significant layers of MaxPooling2D has shown striking improvement in the field of image classification. Specifically, they won the second place in ILSVRC2014 reaching 24.7% of top1 and 7.5% of top5 test error. [8]

2.1.3 ResNet

He Kaiming et al. in [9], considered that training a model based on the mapping error from the input to the output of a neural network is more effective in terms of algorithm convergence so that the model can be deepened as the size of the convolutional layers increases, avoiding the phenomenon of exploding gradients. Specifically, it was assumed that enhancing the depth of a convolutional neural network would enhance the feature extraction process, thereby benefiting the downstream task.

However, with the increased size of the network, the complexity of the objective function to optimize also increases. Mathematically, if the desired underlying mapping is $H(x)$, rather than directly trying to learn $H(x)$, the model learns

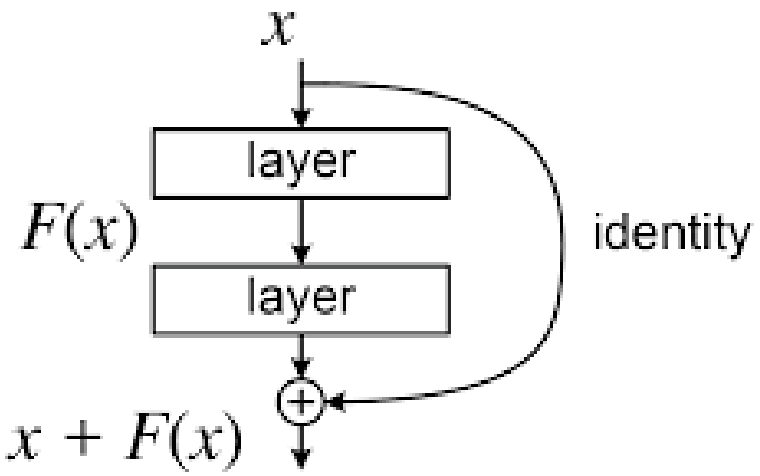


Figure 2.4: ResNet skip connection maps the input of he convolutional layer with an identity mapping which is later added to the output of the layer [9].

$F(x) = H(x) - x$, which is equivalent to $H(x)$. However, it is thought that learning this function is easier than the unreferenced mapping of $F(x)$. In this case, the right side arrow in fig 2.4 represents the so-called *skip connection* which simply maps the input with an identity function and then sums it to the output of the *residual block*. This intuition led them to win first place in the classification competition of ILSVRC2015 and also proved to have noteworthy generalization capabilities for other kinds of computer vision tasks such that it is the current most used backbone for several tasks.

In the contemporary development of such a backbone, skip connections have a somewhat more complex mapping function, as illustrated in [10]. Nonetheless, the meaning behind it is preserved, i.e. if the optimal function is closer to an identity mapping rather than a zero mapping, it may be simpler for the solver to detect perturbations with respect to an identity mapping than to learn the function as completely new.

2.1.4 MobileNet

MobileNet is a family of convolutional neural networks that tries to optimize the memory efficiency of the model and the computational cost of the whole process of training and inference. To do so, in [11] Andrew G. Howard et al. introduced 2 new concepts of convolution which resulted to be significant for the optimization of the computational cost.

Their intuition consisted of decomposing the process of the standard convolution

into 2 steps which are:

- *depthwise convolution*: apply $\#filters = input\_channels$ with variable kernel sizes. By doing so, spatial dimensions are reduced while depth remains the same.
- *pointwise convolution*: apply $\#filters > input\_channels$ of size 1×1. By doing so, spatial dimensions remain the same while the number of channels of the feature maps increases.

Overall the reduction of the cost computation is:

$$\frac{1}{N} + \frac{1}{D_K^2} \tag{2.2}$$

where D_K is the kernel size in the case of depthwise convolution and N is the number of channels in the case of pointwise convolution.

Eventually, in addition to these optimization techniques, they used also other 2 already existing strategies which consist in a multiplier for the number of filters α and a multiplier for the spacial dimensions of the feature maps at each layer ρ. They both range in (0,1].

With these optimizations taken to their limits, they have exhibited a significantly high ratio of model optimization to accuracy degradation, demonstrating the validity of their model.

2.1.5 Collaborative Intelligence

The more research in this area goes on, the more neural network models require a computational cost that from edge devices such as a Jetson Nano are unsustainable given the number of weights in the statistical model, which is in the order of MB. As explained in section 2.1.3, it has been proved that the deeper the model, the more accurate the feature extraction becomes, hence improving the accuracy of the head model prediction [9, 8, 12, 13]. But increasing the depth means adding a lot of convolutions. If we, for example, consider ResNet50 backbone, we can estimate its model size to be 98 MB, which corresponds to 25.6 million weights which is quite impressive.

Based on these assumptions one of the most recent noteworthy research is to find a way to optimize the model size or to find the best strategy to deploy as much as possible the workload to the cloud systems which potentially is assumed to have an infinite computational power.

Recently, Edge Computing (EC) has tried to meet these kinds of needs, but another challenge to address arises which is the balance between latency and computation. Certainly, mobile devices such as drones or rescue systems require a fairly

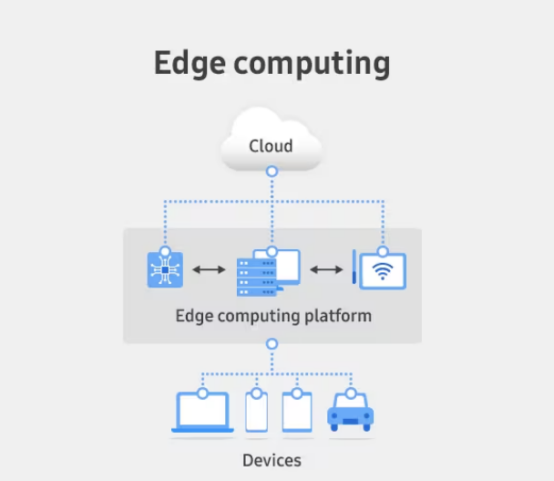


Figure 2.5: Edge computing servers are kept as closer as possible to the mobile device in order to make the communication faster, [Source].

high response speed from the model, which is fully executed in the cloud. At the same time, these systems could be sending information with really high frequency, which could slow down the communication system or even create bottlenecks by exceeding the bandwidth. Therefore, the focus of the optimization, in this case, is communication with the cloud. The options explored were high-bandwidth wireless links such as Long Range, which has a maximum data rate of 37.5 kbps due to duty cycle limitations. Nonetheless, the research conducted by Jiménez Mateo et al. in [14] and Zhang et al. in [15] revealed that despite the wide spectrum available in the mmWave bands, these communication paths suffer from limited throughput due to interferences causing delays and blockages.

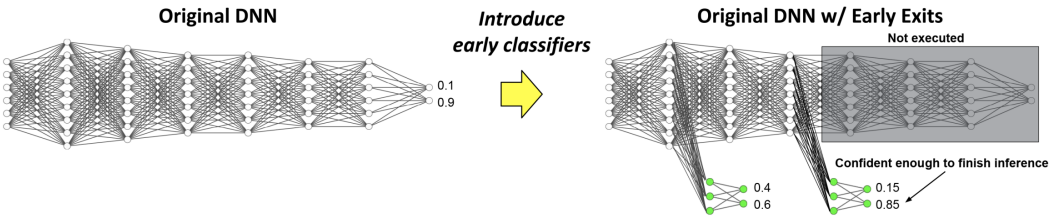


Figure 2.6: The optimal early exit point is found by optimizing the accuracy of a classifier that uses the feature extracted at different stages of the model. [16]

The Early Exiting (EE) strategy, originally introduced by Teerapittayanon et al. [17], represents a cutting-edge advancement in DNN algorithm optimization.

Through the strategic placement of early termination points in the network architecture, EE enables an early conclusion of computations once a desired confidence level is reached for a given input sample. This pioneering approach removes the necessity to reduce the size of DNN models, resulting in a notable reduction in computational complexity. As a result, employing EE technology greatly reduces both the total inference time and computational costs.

This methodology is particularly important in the field of Computer Vision, where varying degrees of sample complexity are present. Notably, EE approaches have been successfully adapted to encompass a range of situations, including mobile-edge-cloud computing systems. Here, neural models are allocated in a way that is dynamically tailored to the computational capabilities of each device, with the aim of optimizing both confidence levels and computational resources. The Split

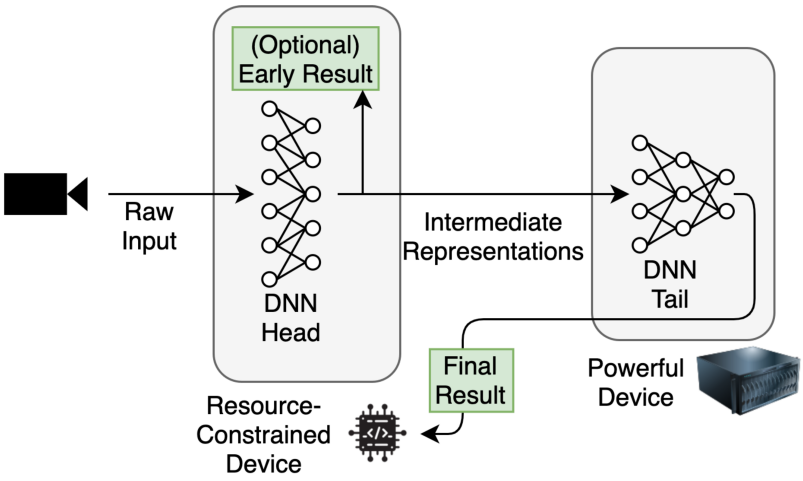


Figure 2.7: Forward pass of the DNN such that it is partly executed on the resource-constrained device and partly in the cloud [18]

Computing technique lies between the other two, as it consists of splitting the forward pass of the model into 2 stages:

1. The first one is executed with the available resources of the mobile device.
2. The second stage instead is computed on the cloud.

and eventually, the output of the required prediction is sent to the cloud server. This ranking is due to the fact that the features extracted by the first convolutional layers that form the *head* are called *high-level features* because they are semantically important for the general understanding of the objects in the image. On the other hand, the features extracted from the *tail* are called *low-level features*, which,

losing the spatial cognition of the elements, are able to extract details such as contours, edges and angles. A high-level feature map typically contains a lower number of channels and, conversely, a low-level feature map contains a higher number of channels, this means that the computational cost is higher the more the sample batch is closer to the tail of the feature extractor because the number of FLOPS increases significantly. As the optimal solution becomes more complete and complex, the number of problems to address also increases. Therefore, the position of the "split point" is a critical topic, clearly discussed by Li et al. in [19], in terms of the available resources on the mobile device and the available bandwidth, showing that the best-performing configuration, presents the *split point* close to the tail of the model.

To the best of our knowledge, split computing is one of the most exceptional frameworks on which most efficient mobile applications of DNN are built. Over recent years, various enhancements have been implemented to minimize model accuracy degradation while simultaneously reducing the volume of data transmitted to the cloud for the forward process.

In the context of techniques that aim to deploy the workload to the cloud (partly or totally), the most significant obstacle is the volume of data represented by feature maps that must be transferred to the cloud for the model to perform the inference process. The approaches outlined below prioritize this optimization challenge.

Bottleneck injection and bit quantization

The *bottleneck* is a special category of layer or block of layers which drastically decreases the number of neurons with respect to the ones of the previous layer. To the best of our knowledge, this concept was originally introduced by Sandler et al. in [11]. it is emphasized that the Rectified Linear Unit (ReLU) activation function can lead to a loss of information in a channel, potentially causing a collapse. However, if there are numerous channels and a structured activation manifold, some information may still be retained in other channels. The paper demonstrates that if the input manifold can be embedded into a considerably lower-dimensional subspace of the activation space, the ReLU transformation can preserve information while introducing necessary complexity into the set of expressible functions. Assuming the manifold of interest is low-dimensional, the optimization of existing neural networks can be addressed by incorporating linear bottleneck layers into the convolutional blocks. Experimental evidence underscores the importance of using linear layers to prevent non-linearities from causing excessive information loss.

In addition to the *artificial bottleneck injection* further data minimization can be performed by using *Bottleneck Quantization*, it is a technique for performing computations and storing tensors at lower bit widths than floating-point precision.

A quantized feature map allows lightening operations of the following layer with lessened precision relative to full precision (floating point) values. For example, quantization can approximate parameters expressed in Float32 (which is the typical representation of neurons and parameters) to Int8 which means reducing the computation time 4 times given the faster execution of the operation between Ints rather than Float values and the same multiplier holds for the model size which is significantly reduced.

The most common quantization strategies used in DNN applications are:

- *Dynamic Quantization*: This procedure entails not only converting the weights to int8 - as is the case in all quantization types - but also changing the activations to int8 immediately before conducting the computation (and hence "dynamic").
- *Post-Training Static Quantization*: It involves feeding batches of data through the network first, and computing distributions of the different activations. This is achieved through inserting "observer" modules at various points in the network that record these distributions. This information is utilized to determine the level of quantization required for different activations during inference.
- *Quantization Aware Training*: It yields the highest accuracy among the three methods. During both forward and backward passes of training with QAT, all weights and activations are "fake quantized", meaning that float values are rounded to imitate int8 values. However, all computations are still performed with floating-point numbers. Therefore, all weight adjustments made during training are conscious of the fact that the model will ultimately be quantized. As a result, this method typically yields greater accuracy compared to the other two methods once quantized.

In particular, bottleneck quantization is a special case of *Dynamic Quantization* and *Quantization Aware Training*.

Knowledge distillation

In machine learning the *knowledge distillation process* is based on the idea of transferring the knowledge from one model to another by comparing the output of the two and setting up the training phase such that so-called *student model* imitates the behavior of the *teacher model*. Let us consider a pretrained model T and a non-pretrained model S but with a more compressed architecture. The task to which a distillation process framework aims, is to use output coming from T as a reference and train S with a loss based on the distance between the reference output $\hat{y}(x|t)$ and the output of the distilled model $y(x|t)$ taking into account, if

possible, also the ground truth $\bar{y}$. Generally, it is used the *Cross-Entropy loss* that has the following expression:

$$E(x|t) = -t^2 \sum_i \hat{y}_i(x|t) \log y_i(x|t) - \sum_i \hat{y}_i \log y_i(x|1) \tag{2.3}$$

where the *temperature factor* t ranges in $[0; \infty]$ and it weights the contribution of the distillation process and x is the input *sample batch*.

In [20] it is demonstrated that knowledge distillation is a special case of model compression. This is possible under the assumption that the logits have zero mean.

Supervised Compression for Split Computing

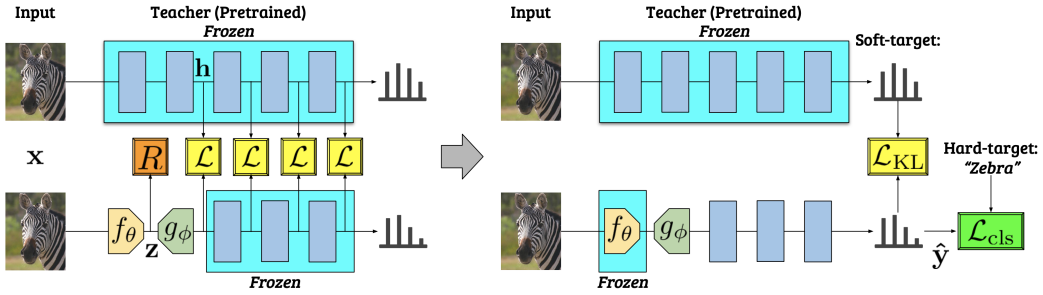


Figure 2.8: Effects of hardware level fault injection on the software performances

In [1] Matsubara et al. merged all the aforementioned techniques and in fig. 2.8 is reported the final architecture of the student S and teacher T models. We can notice that the actual split of the model occurs in S and T is totally transferred to the cloud in order to completely offload the model size of T which is heavier since it is pre-trained without any optimization process. On the other hand, S has the same architecture of T but one layer or a block of layers is replaced with an encoder-decoder structure where a bottleneck layer is injected such that the number of channels significantly decreases and, in addition to that, it performs a bit quantization to Int8 format. The bottleneck layer represents the real split point which means that, during the deployment, the feature maps extracted until this layer will be transferred to the cloud. Eventually, the inference process continues and, in case online training is performed, the outputs of the layer of the teacher corresponding to the replaced one and the outputs of the decoder are compared in order to keep training it.

In this setup the layers that remain the same between T and S are pre-trained, in fact the aim of their training pipeline is to make the model learn to encode and then decode the feature extracted until their point which means basically train

only the layers which for the encoder-decoder structure. Such training phase is divided in two main stages:

1. Training the student model with targets h and tail architecture obtained from the teacher.

2. Fine-tuning the decoder and tail portion with the fixed encoder.

A student model trained through knowledge distillation can be represented as a deterministic two-step mapping, where $z = \lfloor f_\theta(x) \rceil$ and $\hat{h} = g_\phi(z)$, the encoder and decoder mapping respectively, with $\hat{h}(\approx h)$ now being a decompressed intermediate hidden feature in the final student model (refer to Figure 2). Assuming $p_{\psi_j}(y_j|\hat{h})$ represents the output probability distribution of the student model with parameters ψ_j, the fine-tuning step consists in optimizing:

$$\psi_j^* = \arg\min_{\psi_j} -\mathbb{E}_{(x,y)\sim D}[p_{\psi_j}(y_j|g_\phi(\lfloor f_\theta(x) \rceil))] \tag{2.4}$$

This approach reduces the computational load on lower-powered mobile devices by transferring the majority of the processing workload to a high-capacity cloud or edge server. Additionally, the solitary encoder in our entropic student has the capability to accommodate multiple downstream tasks. The findings indicate an improvement in supervised rate-distortion performance for three distinct vision tasks, along with decreased prediction latency end-to-end when compared to multiple baseline methods for neural image compression and feature compression.

2.2 Testing and Reliability

As the digital landscape expands and becomes more integrated into our everyday lives, the necessity for reliable hardware, and consequently software systems, exponentially increases. A hardware-level failure may occur when a logical element becomes stuck at either one or zero. Evaluating the robustness of such a system involves intentionally introducing failures by trying to cover all possible cases, may leading to large search space. After analyzing the obtained results, the most sensitive parts of the hardware, or the errors that most often degrade the reliability of the hardware, are evaluated, and engineers try to adopt the least invasive methods possible to prevent or possibly make the system more robust to these types of errors. Such systems can be hardened at two levels:

- *Hardware level*: for example, by strengthening circuit parts using a different and more resistant material or technology.

- *Software level*: for example, design the program such that if a fault occurs, the behavior of the system remains the same.

Regardless of whether or not a hardware fault may be considered irrelevant to the experiment's goals, it could still have significant consequences on the experiment's outcomes. Thus, it is crucial to highlight the role of fault injection systems in enhancing the resilience of our System Under Testing.

The focus of this thesis work is the *Software level* simulation which involves the reproduction of the effects of hardware aware faults, when forwarding data through a DNN. Nowadays, DNN algorithms exploit the ability of GPUs to support data and thread-level parallelism. In such applications, the impact of permanent faults, potentially due to aging, becomes a significant concern, especially in applications where the expected lifetime of GPUs exceeds ten years.

Historically, GPUs have been known for their susceptibility to transient faults. The sensitivity of GPUs and parallel applications, including CNNs, to radiation and other sources of transient faults has been extensively researched. Shafique et al. in [21, 22] state that such faults can corrupt the output of a running application, leading to potentially catastrophic outcomes in safety-critical domains, and consequently propose an overview of the cutting-edge strategies for different categories of faults. However, as GPUs find applications in sectors like automotive, robotics, aerospace, and health care, where the device's life expectancy ranges from 5 to 10 years, new challenges in GPU reliability evaluation emerge. This life expectancy is notably longer than the typical 1-2 years for GPUs used in gaming, mining, or high-performance computing applications. Consequently, considerations related to aging, degradation, and wear-out effects become crucial as they can lead to permanent hardware faults in GPUs. Such faults, when occurring in safety-critical domains, can produce unacceptable critical effects.

Despite the known vulnerabilities of GPUs to transient faults, there remains a significant gap in understanding the impact of permanent faults on GPUs, especially when running CNNs (Condia et al., [23]). The probability of a permanent fault leading to a critical failure is influenced by both the architecture of the GPU and the characteristics of the software implementing the CNN. An exhaustive gate-level fault simulation for this purpose is impractical due to the high computational requirements. The inherent parallelism of GPU architecture, combined with the sheer number of potential fault points and the complexity of CNN software implementations, makes the evaluation of permanent fault effects extremely challenging.

2.2.1 Key Features of Fault Injection Systems

In the field of computer science, fault injection serves as a testing technique to better comprehend the responses of a Systems Under Testing (SUT) under unusual conditions. This methodology induces stress on the systems in question to evaluate their behavior and identify any latent errors. Data are collected such that statistical

inferences can be performed to find any pattern leading to critical faults based on the characteristics of the injected fault. It is based on the basic concepts of: *faults* (an abnormal physical condition of the system), *errors* (a variation in the behavior of the system with respect to internal signals), and *failure* (a visible change in the behavior of the output system.). Once the critical faults are detected the next

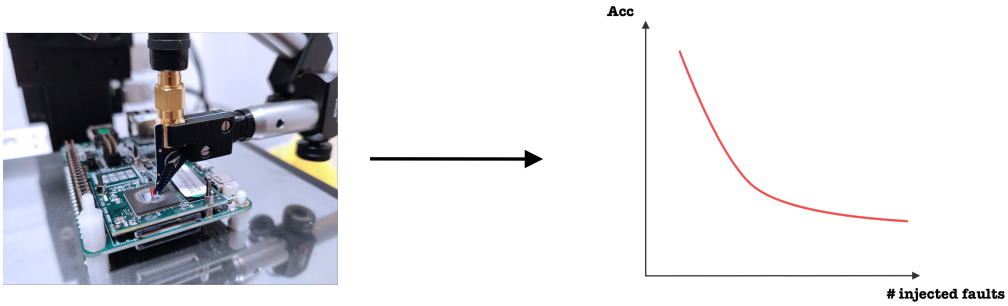


Figure 2.9: Effects of hardware level fault injection on the software performances. [Source]

step is to relate these faults with the sensible parts of the system that are affected and then try to protect them such that they become less sensible to specific faults. Generally, a FI model includes the following components:

- **fault injector**: it injects faults in the target system.
- **workload generator**: defines the basic instructions to exercise the target software during the experiments.
- **controller**: controls the experiment.
- **monitor**: tracks the execution of the workflow of the experiment.
- **data collector**: it performs online data collection.
- **data analyzer**: it processes and analyzes data which can happen also offline.

As Benso et al. shows in [24], depending on the level of injection and the methodologies, FI can be characterized as:

- **Simulation based**: A testing environment is created that simulates the execution of the fault-free system and then the faulty system by comparing the results. Such a simulation can be performed on different levels depending on the feature of the system the experiment is designed to evaluate:
  - *Software level*: high level FI covers different fault categories, such as injecting code to simulate scenarios like buffer overflows or incorrect input

handling, as well as environmental faults that involve manipulating the execution environment, such as changing system parameters or introducing network latency disregarding the hardware features of the resources.

  - *Hardware level*: low level FI which includes the application of high voltages, extreme temperatures, and electromagnetic pulses to electronic components, such as computer memory and Central Processing Units. By subjecting components to conditions beyond their intended operating limits, computer systems can be forced to mis-execute instructions and corrupt critical data.

- **Platform based**: Injection, collection of data, and analysis are performed directly on a physical device which emulates the execution of the SUT.

- **Radiation based**: This can be considered a branch of the platform-based FI, but the nature of the fault changes. Accelerated radiation tests are carried out in order to reproduce an external electromagnetic interference.

It is noteworthy to discuss the feasibility of different FI methods depending on their characteristics, by looking at *cost*, *development effort*, *exactness*, *observability*, *repeatability* and various other metrics of comparison. The detailed study of Ruospo et al. in [25] highlights that the costs to be incurred to carry out experiments are definitively higher in *radiation-based tests* because they irreparably degrade the SUT, given the high amount of radiation to which the devices are exposed and this also justifies the low degree of repeatability of the experiments. Nevertheless, the highest workload depends on the level on which you are working because the developer has to build build, and control all the components of the FI environment. Hence, as the application's level decreases, the development effort increases, and the *platform-oriented tests* are conducted directly on the physical device. "Exactness" follows the same trend because it relies on how realistically the framework reproduces the defects; therefore, the lower the level, the more accurate must be the application. From an observability perspective, it is quite intuitive that the *simulation-based level* outperforms all others since the infrastructure of the application by itself makes you custom the capability of the system to collect data and compute the corresponding metrics.

Some further detailed characterization can be described from the faults' point of view. Depending on the duration of the fault, a preliminary classification is:

- *Permanent* if they are continuous and stable over time, typically resulting from irreversible physical damage.

- *Transient* if they are transient in nature and frequently arise due to external disruptions.

- *Intermittent faults*: if they occur within a specified time.

On the other hand, the faults can be also classified with respect to their nature:

- **Power-related faults**: Power source problems, such as varying voltages or sudden power spikes.
- **Overheating**: Elevated temperatures can result in reduced performance or even trigger the accelerator to cease functioning in order to safeguard itself.
- **Wear and Tear**: Extended utilization of AI accelerators can result in the deterioration of hardware elements, including connectors, sockets, or solder joints. These physical issues can lead to a decline in performance or eventual malfunction over time.

Chapter 3

Related Work

The current fault injection methodology has a varied range of approaches, each designed to expose system vulnerabilities. This chapter presents an extensive review of available fault injection techniques, emphasizing their relevance and efficiency in the context of AI-augmented IoT systems. The following segments explore specific fault injection methodologies, providing a detailed examination of their principles, applications, and results. The primary objective of this chapter is to establish a framework for evaluating and selecting fault injection techniques that are best suited to the unique requirements of AI-powered IoT systems, by synthesizing insights from existing research.

3.1 Fault Injection for DNN models robustness assessment

As discussed in Chapter (2), to ensure proper training of DNNs, a large amount of data is required resulting in a high number of operations (e.g. scalar product, activation functions and derivatives for gradient computation).

Despite the low computational cost of each operation, the complex architectures involved necessitate significant computational power. The nesting of multiple functions implies a complex optimization process leading to high overall computational costs. Consequently, similarly intricate hardware infrastructure, such as embedded GPU accelerators, are employed. GPUs execute numerous parallel processes, allowing for optimal efficiency and if a component fails the model's performance can be significantly impacted or not.

In this testing environment, two main components can be reinforced:

- *Hardware*: by replacing the most critical components with the same product made of a different material or protecting the part to avoid overload.

- *Software*: by simulating the effect of a hardware fault on the DNN models by purposely injecting random noise into the software's basic elements (i.e. weights and neurons). Then, based on the results of analysis aimed at extracting the critical fault characteristics, the model's architecture can be reinforced or a Fault Aware Training (FAT) can be carried out.

Clearly, the most expensive and time-consuming solution to implement is hardware hardening, as it may rely on destructive testing or involve refactoring a new hardware solution with more expensive materials. Thus, this chapter will delve into the literature on fault detection and software solutions.

Some studies in this area, such as Ruospo et al. in [25], focus on the properties of the characterization of FI techniques and give an overview of the challenges that an FI system has to face today. They then explain the pros and cons of all these metodologies, pointing out that simulation-based FIs at the software level are cheaper, faster, more controllable and easier to implement. In particular, in these simulations, the SUT is the probabilistic model and, thanks to the tools provided by Pytorch and Tensorflow[1], it is easier for the developer to perturb some internal features of the model at inference time. In this process custom fault injection functions can be used in order to compute appropriate metrics for error evaluation. On the other hand, the software-level fault injections may not faithfully reproduce the real effects of faults due to a lack of information about the hardware used, then they may lose accuracy in the real case representation.

Hardware-level FI frameworks are employed to overcome this problem and to work at a lower level of abstraction which assures a higher injection accuracy. Generally, the choice of one or another abstraction level depends on the time consumption that an experiment would take and on the economic resources at the disposal of the developer, because Hardware FIs are more time-consuming and less cost-effective, an attempt of catching hardware DNNs vulnerabilities is presented in [26]. Thus, when a FI framework is designed, there are 2 main features to take into account: the *coverage level* of the real case simulation and the *computational effort* of the experiments.

3.1.1 Fault Injection framworks

Ares [27] is one of the first software-level FI tools aiming at understanding the relationship between fault rate and model accuracy, therefore focusing on the *coverage level* rather than efficiency.

Ares presents an innovative approach to fault analysis in the DNN model, by directly injecting faults into designated points, such as weights, activations, and

[1]Main Python libraries for Deep Learning models development

hidden states. This application-based perspective enables a detailed investigation of fault behavior, leading to a more sophisticated understanding of fault tolerance across different components of the network. Additionally, the paper emphasises that more precies data types for models greatly boost resilience (up to 10×), providing a new aspect to enhancement plans.

In order to assess Ares's reliability as a Fault injector, it has been tested on a real DNN accelerator, exhibiting its precision in capturing the bit-error behavior observed in real hardware.

For what concerns the fairness aspect of a simulation, a mathematical estimation of the number of campaigns and the number of faults to inject was provided by Leveugle et al. in [28] in order to reach the desired confidence in the experiment. Assuming the following statement:

1. Features that characterize the soft errors follow a normal distribution
2. During the random sampling of the faults without replacement, a uniform distribution must be shaped
3. The number of faults of the initial population N depends on the number of components that can be perturbed, on the error model and on the number of cycles of workload

it provides a basis for the error level e corresponding to a campaign with a specific number of injected faults n which is:

$$e = t \times \sqrt{\frac{p \times (1-p)}{n} \times \frac{N-n}{N-1}} \tag{3.1}$$

where N is the initial population size, p is the estimated proportion of individuals in the population having a given characteristic and eventually t is the cut-off point of the population distribution corresponding to the confidence level.

During this thesis work, the main infrastructure used to inject faults in DNN at run time is *pytorchFI*, presented in [29]. PytorchFI is a runtime perturbation tool that stands out for its user-friendly approach from the already existing frameworks such as Ares [27] and TensorFI [30]. PyTorchFI excels in terms of runtime overhead, boasting a native implementation that incurs negligible additional computational cost. This efficiency is crucial, particularly for applications that demand real-time or near-real-time processing.

The approaches of use proposed in [29] are described as follows:

1. A simple approach is to add an intermediate layer after each convolutional layer. This layer contains a transformation step to perturb the output values before they are passed to the next layer in the network. However, using

this method to investigate different perturbation models requires significant changes to the network configuration. For deep networks with many layers, or those with custom layers distributed between convolutions, implementing this approach would require significant user effort.

2. Alternatively, one could choose to modify the PyTorch source code to intercept neuron computations for perturbation. However, this approach faces challenges in terms of portability, as it could require different implementations of convolutions for different processing backends such as CPU, GPU and others. It would also require patching of scripts and ongoing developer maintenance to ensure compatibility with future versions of PyTorch.

Ruospo et al. [31] and Zheng et al. [32] contribute significantly to the evolving landscape of research focused on evaluating the reliability of Deep Neural Networks (DNNs) and on the hardening techniques by injecting faults a both hardware and software level. [31] proposed a novel pipelined multi-level fault injector for DNNs, specifically tailored for reducing fault simulation time at the Hardware Description Level (HDL). This approach addresses the challenge of efficiently assessing DNN reliability in the presence of specific hardware architectures. By viewing neural network layers as pipeline stages and synchronizing single inference computations, the framework enables designers to evaluate the suitability and robustness of hardware for DNN-based applications.

In contrast, MindFI [32], which is based on Mindspore [2] provides an additional fault injection strategy which consists in injecting noise in the data, specifically it allows at simulating different faults such as image noise, labeling errors, and cosmic particle-induced bit flips.

Ruospo et al. made an important contribution by developing an additional FI framework called SCI-FI [33]. SCI-FI is comprised of four FI models, two of which perturb the parameters (i.e. the weights) and two that perturb the neurons. The paper examines three strategies for strengthening software, namely, substituting activation functions, employing fault-free model output, and implementing skip connections within the blocks.

Another interesting tool for these kinds of simulations was developed by a team from Harvard University, NVIDIA Corporation, and the University of Illinois in [34] where they proposed a software-directed selective protection technique by optimizing the fault detection. In this paper it is included a Feature-Map

[2]Mindspore is a framework that helps data scientists to design training and inference pipelines specifically in the context of mobile applications. It implements *Source Transmission* automatic differentiation approach whose aim is to make the computation of the gradient at training time as efficient as possible supporting complex control flow scenarios, higher-order functions, and closures. For more details please check the Mindspore documentation.

Level Resilience (FLR) and Inference Level Resilience (ILR), complemented by the combined approach, FILR. These methods aim to achieve comprehensive error coverage with minimal computational overhead. The peculiarity of such approaches stands in the definition of a tailored loss which lets the model learn to pinpoint the most sensible feature maps to protect. It achieves nearly full error coverage while incurring only a modest average runtime overhead of 48%.

One of the first noticeable FI frameworks that reached a fault coverage higher than a random sampling of faults was ISimDL, developed by Colucci et al. and presented in [35]. ISimDL utilizes neuron sensitivity for producing importance sampling-based fault scenarios, presenting an efficient and precise means of identifying critical faults. Furthermore, it supports FAT by selecting faults that result in errors and inserting them during the DNN training process to increase the network's resilience to such errors. ISimDL can achieve precision 15 times higher in detecting critical faults than random sampling and reduce FAT overhead by over 12 times.

The fault injection frameworks outlined in this section have displayed considerable strengths, closely adhering to the requirements set forth by the current state-of-the-art. Their versatility and effectiveness in simulating various fault scenarios underscore their importance in fault tolerance testing. However, it is important to note that these frameworks tend to maintain a certain level of hardware agnosticism, which may be advantageous in certain contexts, yet may also warrant consideration in scenarios where specific hardware nuances play a critical role. This characteristic prompts a thoughtful evaluation of their suitability in specific testing environments.

3.1.2 Hardening techniques

As outlined in [21], in the face of permanent faults, characterized by irreparable chip damage, the most effective recourse often entails the replacement of the faulty chip or component, albeit at a significant cost. A more economically viable alternative involves the selective discarding of only the erroneous bits or bytes of the afflicted component, thereby minimizing incurred expenses. Specifically tailored to ML systems, techniques like fault-aware training, pruning, mapping, and activation clipping play pivotal roles in addressing permanent faults. Fault-aware training entails the training of DNNs to account for various faults at multiple levels, spanning from the transistor to logic levels. While potent, this approach does entail substantial computational overhead. Fault-aware pruning involves the excision of DNN connections and parameters associated with faulty processing elements or nodes, guided by fault maps of the underlying hardware. Fault-aware mapping exploits the saliency of DNN parameters to define a mapping while retaining the salient components. Meanwhile, fault-aware activation clipping involves the capping of activation values surpassing a predefined threshold for fault-free neural networks,

obviating the need for either pruning or retraining.

On the other hand soft errors, arising from transient faults can be mitigated through various techniques. These include interleaving to prevent consecutive bit errors, additional error detection circuitry, periodic scrubbing to remove errors, hardware redundancy with voting mechanisms, and error detection and correction codes. Recent advancements also include replicating hardware accelerators with a majority voting for added safety. These approaches involve trade-offs in terms of error detection, correction capabilities, area, power, and latency. Redundancy-based methods may incur significant area overhead. Additionally, studies highlight that bit-flips from 1 to 0 have a more significant impact on system accuracy in DNN-based systems, informing the choice of error-correction mechanisms for critical bit-flips.

Eventually, it is noteworthy to mention 2 simple yet effective software-reinforcing techniques which showed impressive results. They were proposed by Cavagnero et al. in [36]. The intuition behind their research is that when a fault is injected the most affected layer is generally a Convolution because it contains way more weight and neurons than others (e.g. FC layer), and specifically some of these elements can reach very high values depending on the fault which lead the model to a misclassification.

Thus, they have tried to clip the output feature maps in the range [0,6] using a ReLU6 activation function. By doing so they did not reduce the accuracy of the model because the convolution itself can still output values in a range $[-\infty, \infty]$, but when they are propagated to the next one, they will fall within the desired range.

With the same assumption, another important contribution that they gave to the fault injection environment, was to further change the architecture of the model by swapping the position of the batch-normalization layer with the convolutional layer. The batch normalization standardizes the values of the tensors within the same batch, then it is able to detect the outliers from the actual data distribution. Then huge or very low values in the feature maps will be detected as outliers and then standardized according to the distribution of the current batch. This paper has been an inspiration for the design of our hardening techniques, in the context of fault injection in supervised compression for split computing models.

Chapter 4

Experiment settings

Up to this point, we have outlined the project's motivations, as well as provided a preliminary study aimed at gaining a deep understanding of the theory behind it and previous research. This chapter will outline the fundamental aspects of the research, including a description of the methods used for the analysis of the fault injection results to establish a baseline for evaluation. By combining existing knowledge with our proposed analysis, we have designed effective software reinforcement techniques. In particular, the experiments carried out during the thesis work consist of:

- Fault injecting in the backbone of models that are already available in [1]: ResNet50 Classifier for image classification, Faster RCNN with backbone ResNet50 for the object detection and DeeplabV3 with backbone ResNet50 for the Semantic Segmentation in order to assess the robustness of different configurations.

- Fault injecting in newly trained student models suitable for the task of Image Classification and Object Detection.

- Hardening techniques based on the analysis of results of fault injection in default models.

4.1 Fault injection framework

As already mentioned in Chapter (3), the Fault injector used for the injection campaigns is based on pytorchFI which allows perturbing at inference time a neuron or a weight located in a specific position of the feature map/weights tensor. We propose an extension of the available PytorchFI tool that enables to corrupt the parameters of the CNN as well as the feature map.

4.1.1 Weights FI

This method entails randomly perturbing a value in the weights tensor, which represents the output of the Neural Network's learning process. As the models we are testing were trained using the split computing paradigm and knowledge distillation, they have fewer parameters than their original versions due to model size optimization. This makes them more susceptible to performance issues caused by hardware faults using the stuck-at fault model.

Firstly, it is computed the number of faults to inject n, (Leveugle et al., [28]) which is computed as:

$$n = \frac{N}{1 + e^2 \times \frac{N-1}{t^2 \times p \times (1-p)}} \tag{4.1}$$

Starting from an initial population size N, this resulting total number of faults satisfies an error margin e, the cut-off point t corresponding to a confidence level, and the probability p of randomly picking a specific value of the features describing the fault. When weights are corrupt, the characteristics of a fault are: *kernel* K, *channel* Ch, *row* R, *column* Col which define the exact position of the weight in the parameters tensor to perturb and, in addition to these, another feature which defines the severity of the perturbation is the *bitmask* b.

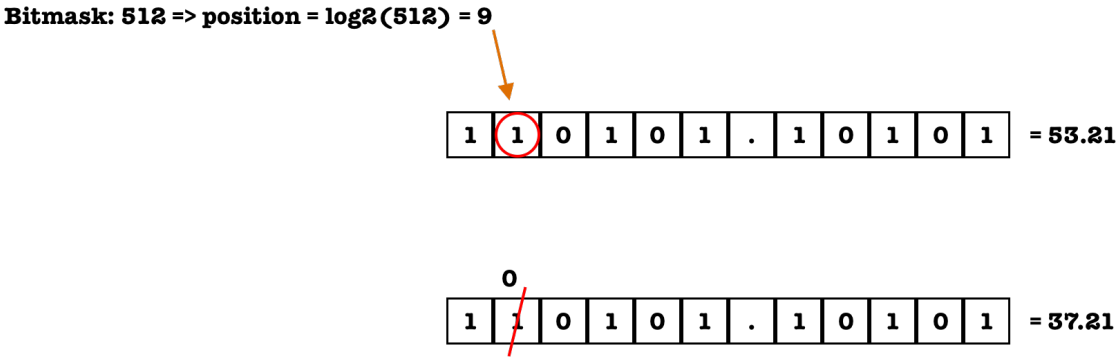


Figure 4.1: Bit-flip example

Since the injection under analysis is based on bit-flips, the bitmask is still a positional feature because it can actually define the position of the bit to flip such that the corresponding real number changes. A toy example of what "bitmask" feature represents is shown in Fig 4.1. If we consider a real number expressed as a 32-bit Single-Precision Floating-Point number, it can be easily converted to a binary representation, and depending on the perturbed bit position, it can be more or less affected by the injection.

(K,Ch,R,Col) is a vector of integers and one of the assumptions behind Eq 4.1 requires that:x

$$K \sim Uniform(0, shape[0]), \tag{4.2}$$
$$Ch \sim Uniform(0, shape[1]), \tag{4.3}$$
$$R \sim Uniform(0, shape[2]), \tag{4.4}$$
$$Col \sim Uniform(0, shape[3]) \tag{4.5}$$

where the components of the vector *shape* are the dimensions of the kernel tensor to corrupt in case the injection occurs in a convolutional layer, otherwise if the corrupted layer is a dense layer, *shape* vector will have only 2 components (R and Col) due to the shape of the corresponding weights tensor. For what concerns instead b, by hypothesis, it still must follow a Uniform distribution but the range of random integer sampling is bounded between the Lowest Significant Bit (LSB) = 19 and Most Significant Bit (MSB) = 31 because of tensors data type. Since bit corruption in LSB typically induces no effect on the DNN, most of the time results are masked.

4.1.2 Neuron FI

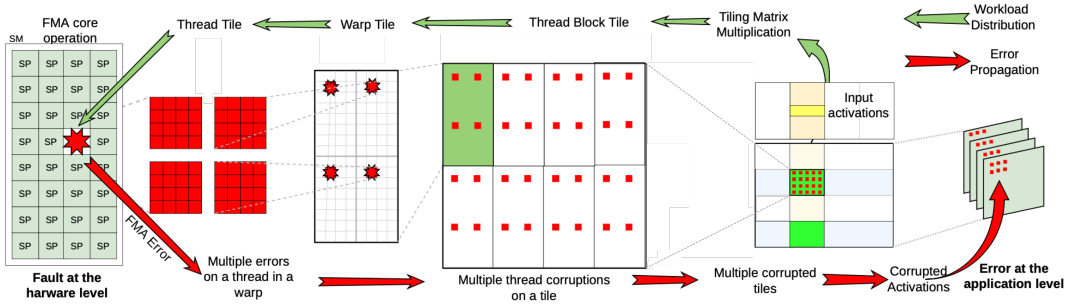


Figure 4.2: Hardware-aware fault propagation based on GPU's GeMM algorithm workload distribution.

As described in Section 2.1.5, the *bottleneck layer* precedes the *split point* of the DNN architecture and its aim is to drastically decrease the number of channels of the output feature map. By doing so, the number of neurons of the corresponding layer decreases and then it increases the probability that the model's robustness is weakened by a hardware-level fault which makes it interesting to test such models with neuron-level injection. This section presents the process of fault injection at *Neuron level* which accepts the following hyperparameters:

- *layer_start* and *layer_stop*: to better simulate error propagation across successive feature maps and the convolutional layer, the framework allows for the specification of the layer range for the injection to occur.
- *num_threads*: number of threads to corrupt per Streaming Multiprocessor.
- *tail_size*: the size of the threads block tile.
- *block_fault_rate*: portion of tiles executed in a faulty Streaming Multiprocessor.
- *neuron_fault_rate*: portion of affected neurons due to the fault propagation.

Typically the Convolutional and Fully-Connected are mapped into GPUs as General Matrix Multiplication(GeMM). Such implementation of fault injection relies on the GeMM algorithms that are implemented in the GPUs, as shown in fig 4.2. The matrix is divided into sub-matrices called *tiles*, which are distributed among the parallel cores. We took inspiration from the fault propagation that may occur at hardware level. More precisely, if one Streaming Multiprocessor (SM) has a defective Fused Multiply–Add (FMA) core, the resulting error may propagate to one or more threads per warp, causing multiple data corruption that is also distributed at the output of the tile. Consequently, processing more than one tile on the malfunctioning SM produces a consequent effect on its respective tiles' outcomes. On the basis of this knowledge, we simulated the behavior of the algorithm for the MM when some hardware components are damaged.

Thus, in this particular fault injection scenario, the list of faults comprises all feasible combinations of the aforementioned hyperparameters and these hyperparameters are preliminarily set in accordance with the hardware architecture that is available to the researcher.

Once the fault list is properly generated the evaluation process is performed: generally, the predictions of the *Corrupted Models* are compared with *Golden Model* predictions.

Before going in-depth into the detailed description of the different setups it is noteworthy to point out that the injection will always occur within the first layers of the bottleneck because the only tensors of layer parameters that are still trainable also during the fine-tuning step of SC2, belong to the encoder structure of the bottleneck.

4.2 Reliability Evaluation of Image classification application

In the dynamic and ever-evolving field of computer vision, image classification serves as a crucial task that demonstrates the profound influence of convolutional neural

networks (CNNs) on visual recognition. The paper [37] by Peng et al., presents a detailed investigation of cutting-edge models that have not only transformed image classification but have also promoted advancements in numerous computer vision applications.

Image classification involves categorizing images into predefined classes. It is a fundamental building block for numerous downstream tasks, such as object detection, scene understanding, and facial recognition.

Commonly utilized metrics for the assessment of image classification models include:

- *Accuracy*: the ratio of correctly classified instances to the total number of instances in the dataset. It provides a fundamental measure of overall model performance.
- *Precision and Recall*: have particular relevance to imbalanced datasets. Precision emphasizes the accuracy of positive predictions while recall emphasies the model's ability to capture all relevant instances of a particular class.
- *F1-Score*: The harmonic mean of precision and recall, offering a balanced assessment of classification performance.
- *Top-k Accuracy*: Extending beyond binary classification, this metric evaluates whether the correct class is within the top-k predicted classes, reflecting a broader understanding of the model's proficiency. Such a metric extension can be applied also to the metrics of *precision*, *recall* and *F1-score*.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{4.6}$$

$$\text{Recall} = \frac{TP}{TP + FN}, \tag{4.7}$$

$$\text{Precision} = \frac{TP}{TP + FP}, \tag{4.8}$$

$$\text{F1-Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.9}$$

These metrics are used in order to evaluate the *Faulty Model* by taking as reference the *Golden Model* guesses such that the corrupted predictions can be labeled as *Critical*[1], *SDC*[2] or *Masked*[3].

[1]when the injected error induces a misclassification

[2]Silent Data Corruption without prediction changes

[3]when the injected error does not change the inference outputs

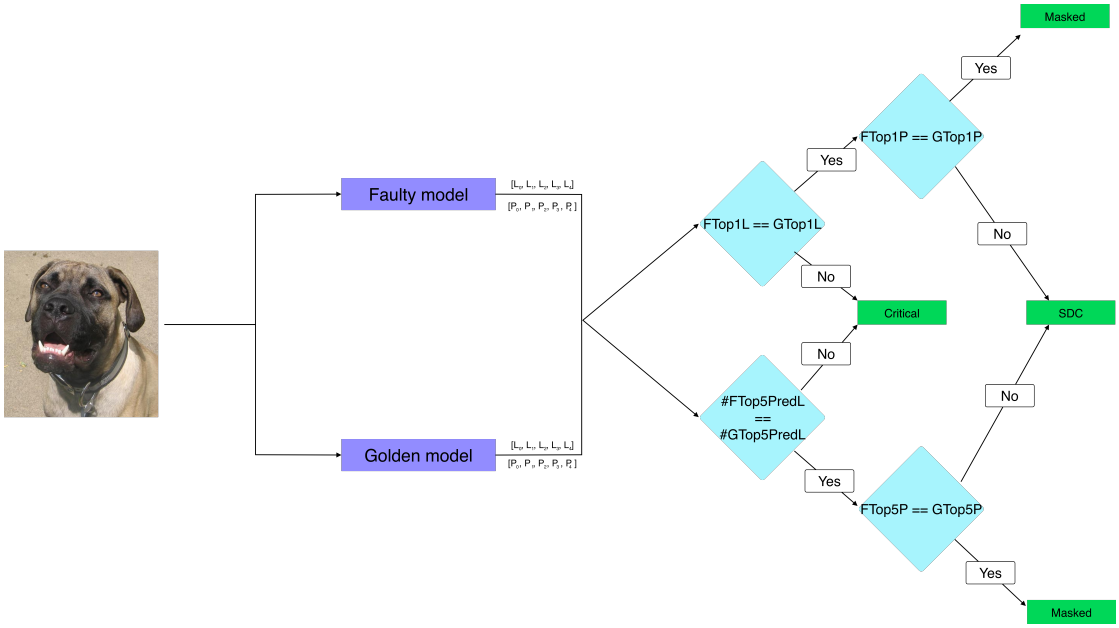


Figure 4.3: Corrupted prediction evaluation process for image classification models

Fig. 4.3 outlines the evaluation process used in every FI in image classification task models. For each test image, only correct predictions of the golden model and correct predictions of the corrupted models are considered (w.r.t. the ground truth). Specifically, the Top 5 predictions, ordered by confidence level, are extracted from the output of each model alongside their corresponding probabilities. If the Top1 label predictions differ, the faulty Top1 prediction is labeled *Critical*, and if the Top5 predictions do not contain the same number of correctly classified labels, the prediction is labeled *Critical*. Otherwise, if the conditions are satisfied, confidence levels are compared, and the labels assigned will be either SDC or *Masked* depending on their equality.

The subsequent sections will detail the environment in which the models were trained in relation to supervised compression for split computing. This will include configurations, loss criteria, and other hyperparameters. Consequently, the setup for the FI campaigns will also be presented, i.e., the injection type of the layers, number of trials, and other hyperparameters that are crucial for achieve a clear overview and comprehending the results of the simulation analysis.

4.2.1 Model: SC ResNet50 Classifier

Most common classification models often rely on feature extractors followed by a classification block that involves one or more Dense layers and a softmax and

together they compute the probability distribution of the class labels referring to the current image. In SC2 framework a standard model for each task was already implemented, specifically, for image classification, it was ResNet50 Classifier. As the name suggests, the model is composed of a ResNet50 backbone, already presented in section 2.1.3, and a classifier block that contains the sequence of AveragePooling, FC layer, and a softmax activation function [1].

The bottleneck layer replaces many of the layers from the original model while avoiding skip connections in order to obtain the student model and at the split point a "SimpleQuantization" is performed on the output feature map, casting the data from $Float32$ into $Int8$ format. At the end of the bottleneck, the Mean Squared Error (MSE) between the output of the teacher and the output of the student until that point is computed. This is utilized in the loss computation, and subsequently, the data is forwarded through the rest of the network. The corresponding output of student and teacher models is used to compute the MSE. For the backward step, the accumulated losses are summed and then employed to update the weights. The images are fed through the network for 20 epochs, with a batch size of 16.

The gradient step on the learnable parameter search space is weighted by the learning rate of 0.001, which is subsequently decreased by a factor of 0.1 every 5 epochs to facilitate model convergence. Eventually, all trained models vary according to the "number of channels" in the output feature map at the split point, which is a hyperparameter that takes discrete values within the discrete interval [1,2,3,6,9,12].

On top of these trained models, we performed a fault injection at the weight level. Specifically, we injected faults in the convolutional layers considering the following hyperparameters:

Layers
cut-off distribution point
confidence level
probability fault instances

Table 4.1: Weight fault injection hyperparameter

Additionally, a neuron-level fault injection is performed whose configuration is described by the hyperparameters of the of the following table:

4.2.2 Model: SC MobileNet Classifier

As discussed in chapter (3), the more reliable the experiment, the greater the number of faults to inject. This, in turn, results in time-consuming campaigns. In

Layers
Trials
size_tail
block_fault_rate
neuron_fault_rate
num_threads

Table 4.2: Neuron fault injection hyperparameter

some instances, the simulator must inject ≈ 6000 faults to achieve a 99% confidence level. Given that the ResNet50 classifier typically takes 4 minutes to evaluate 500 images, the overall campaign becomes extremely time-consuming. Even a small improvement of only 1 second would greatly reduce simulation time. Therefore, a preliminary assessment of the inference time on different models has been carried out before the choice of *MobileNet Classifier* as one of the additional models to implement in the SC2 framework which resulted in being significantly faster than the ResNet50 Classifier due to its reduced model size.

One of the key assumptions of the SC2 framework's training process is that the teacher models have previously been trained on the same dataset. As the models provided by Pytorch have not been pre-trained on the dataset we intend to use, we have designed an appropriate pipeline to train *MobileNet* on *Cifar100*.

First and foremost, for the sake of model size efficiency, *MobileNet v3 small* have been chosen as the target version which is both stable and is actually used for deployment in real hardware.

The innovation of MobileNet v3 was the introduction of a selection of layers of the previous versions as building blocks. These layers are complemented by updated *swish* nonlinearities. The *sigmoid* is used by both the squeeze and excitation components, as well as the swish nonlinearity, which can result in computational inefficiencies and fixed point arithmetic accuracy issues. Their solution is to substitute the sigmoid with the hard sigmoid (Howard et al., [38]). A small version of such a model differs from the large one because of the employment of less and more lightweight convolutional layers which decrease the model size and consequently speed up the inference.

For what concerns the learning pipeline, the most suitable criterion of loss for an image classification task is the *Cross-Entropy loss*, whose expression for multivariate classification is:

$$H(y_i, \hat{y}_i) = -\sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij}) - (1 - y_{ij}) \log(1 - \hat{y}_{ij}) \tag{4.10}$$

where

- C is the cardinality of the set of classes
- $y_{i,j}$ is an indicator variable which results 1 if i belongs to the class j, otherwise it results 0
- $\hat{y}_{i,j}$ is the predicted probability by the algorithm that the label i belongs to the class j.

Once the loss is computed, the optimizers' aim is to record it and then update the weights according to the algorithm, specifically, two algorithms have been considered:

- Stochastic Gradient Descent (SGD): Computes the derivatives of the optimization function only with respect to batches of the dataset, such that, the gradient step is an average among all the optimization results. Especially when dealing with large datasets, makes the algorithm more efficient. Moreover, the randomization of the data introduces some noise which prevents the model not to learning the inner paths in the organization of the data.
- Root Mean Squared Propagation (RMSProp): is designed to adapt the learning rates (γ) of individual model parameters by scaling the gradients based on their historical magnitudes.

Eventually, in order to favor the learning algorithm convergence, it is best practice to scale γ according to a schedule and in this specific case 2 different schedulers are considered: the first one is a *Multi-Step* scheduler that scales γ according to a factor ϵ after every s epochs and the second one is a *Cosine Annealing Warm Up* scheduler that, after the model is trained for a specific number of epochs ($warmup\_epochs$), γ is scaled according to the following schedule:

$$\eta_t = \eta^i_{min} + \frac{1}{2}(\eta^i_{max} - \eta^i_{min}) \left(1 + \cos\left(\frac{T_{curr}}{T_i}\pi\right)\right) \tag{4.11}$$

To sum up, the hyperparameters tuned for the training step of MobilenetV3Small are reported in table 4.3

Once the model was trained with the best configuration, we used it as a teacher model, and then a proper encoder-decoder structure was designed in order to cover the 2^{nd},3^{rd} and 4^{th} layers of the original model, by reducing the initial 15 convolutions to 5 convolutions, avoiding the inverted skip connections and also SqueezeExcitation modules belonging to the original model. The main challenge to be faced during the design phase of the bottleneck was the matching of the tensor shapes. In general, in order to avoid any software error in the forward function of a CNN algorithm, it is only needed to match the number of channels of the input and output feature maps with respect to the required number of channels of the

γ
s
$weight\_decay$ (ρ)
ϵ
$batch\_size$
$momentum$
$warmup\_epochs$
num_epochs

Table 4.3: MobileNet training hyperparameter

previous and following convolutional block. Nonetheless, in the SC2 framework, in order to train the newly designed encoder of the bottleneck, it is needed to match the spatial dimensions of the teacher model output feature maps, i.e. output of the bottleneck layer, must match in shape with the output of the last replaced layer. The reason stands behind the computation of the loss (Fig. 4.4). As presented in section 2.1.5 the MSE loss is used to train the bottleneck, thus, in order to keep fairness in the training algorithm, the shapes of the compared tensors must match, and the parameters of the new convolutions must accordingly be tuned. Starting

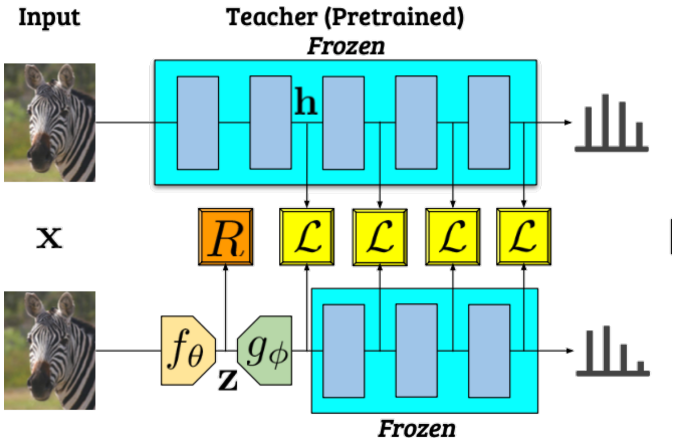


Figure 4.4: Artificial bottleneck training from [1]

from the basic convolution operation, it can be found that given a feature map as input with height H_{in} and applying a 2D convolution with padding p, stride s dilation d and kernel size k, the height of the output feature map is:

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times p[0] - d[0] \times (k[0] - 1) - 1}{s[0]} + 1 \right\rfloor \tag{4.12}$$

The same formula has been used in order to match the width spatial dimension

considering the second components of the vectors describing the convolution:

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times p[1] - d[1] \times (k[1] - 1) - 1}{s[1]} + 1 \right\rfloor \tag{4.13}$$

Indeed, for what concerns the tuning of the third dimension (i.e. convolution number of output channels), the parameters of the internal convolutional blocks, have been found as a trade-off between the head model size and the accuracy of the whole bottleneck to encode and then decode the input feature map.

Thus, under the above-mentioned assumptions, the artificial bottleneck architecture injected in MobileNet is shown in Fig. 4.5

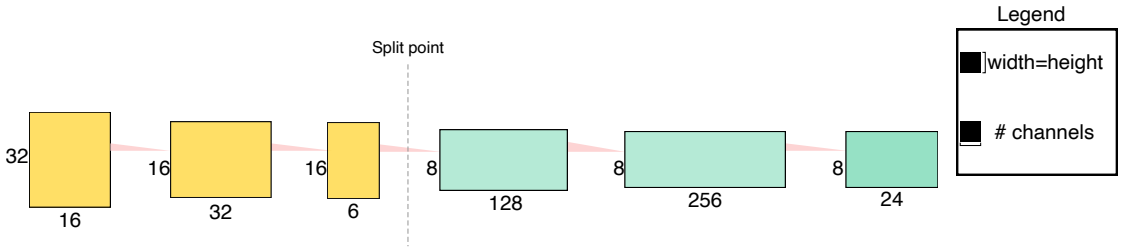


Figure 4.5: Split Mobilenet bottleneck architecture

Once defined the design and training processes of the Artificial Bottleneck block. The student model is submitted to the robustness test with our pytorchFI-based Fault Injector. As the neuron FI results in a time consumption that is twice that of weight FI, it is more beneficial for this particular model to perform only the second one, where the split point outputs 6 channels.

The simulation is characterized by same hyperparameters shown in A.9 and A.8.

4.3 Reliability Evaluation of Object Detection applications

As preannounced in section 2.1.1, among the various activities in this area, *object detection* represents a crucial and demanding challenge. The key to object detection is not only to recognize the presence of objects in an image or video but also to locate them, frequently using bounding boxes. This capability finds extensive applications in various domains: retail and inventory management, geospatial imaging, gesture recognition, quality control and industrial automation.

The importance of object detection lies in its crucial role in enabling machines to understand the visual world in a manner similar to human perception. Rather than just identifying individual objects, object detection allows for a more detailed understanding of complex scenes by recognizing the spatial relationships and

contextual interactions between multiple entities. Such thorough comprehension of information is vital, spanning from recognizing individuals on the street for autonomous cars to recognizing peculiarities in medical photographs [39].

The object detection task is composed of two downstream tasks:

- *Multi-variate Regression* of the coordinates that locate the bounding boxes.
- *Classification* of the boxes that define the detected object.

Figure 4.6: Misprediction for each downstream task. Specifically, the top-most figure represents a failure in coordinates regression and the bot-most figure represents a misclassification. [Source]

Therefore, the failure of one of the two tasks may lead to a misprediction as shown in Fig 4.6, where the red box is predicted by the Golden model and the yellow one is predicted by the corrupted model. In order to better comprehend the taxonomy used in the labeling process of the faulty predictions let us now focus on

Fig 4.7. Let us consider the prediction of the golden model $\hat{\underline{y}}_g$ one at a time, and the vector of faulty predictions $\hat{\mathbf{y}}_f$, each of them is represented by a label $\hat{y}_{(*,l)}$, and the vector containing the coordinates of the bottom-left corner and of the top-right corner of the bounding box $\hat{\underline{y}}_{(*,[x_1,y_1,x_2,y_2])}$. It is computed the distance between the $\hat{\underline{y}}_{(g,[x_1,y_1,x_2,y_2])}$ and $\hat{y}_{(f,[x_1,y_1,x_2,y_2])}$ and the faulty prediction which minimizes this distance is considered as the corrupted version of the $\hat{\underline{y}}_g$ under analysis. Once the candidate has been found, the faulty bounding box is classified as critical if $IoU <= 60\%$, otherwise, the correspondence of the label is checked, in case it is not verified the box is labeled as *Critical*, otherwise, it labeled as *SDC* or *Covered* according to some arbitrary thresholds of IoU level.

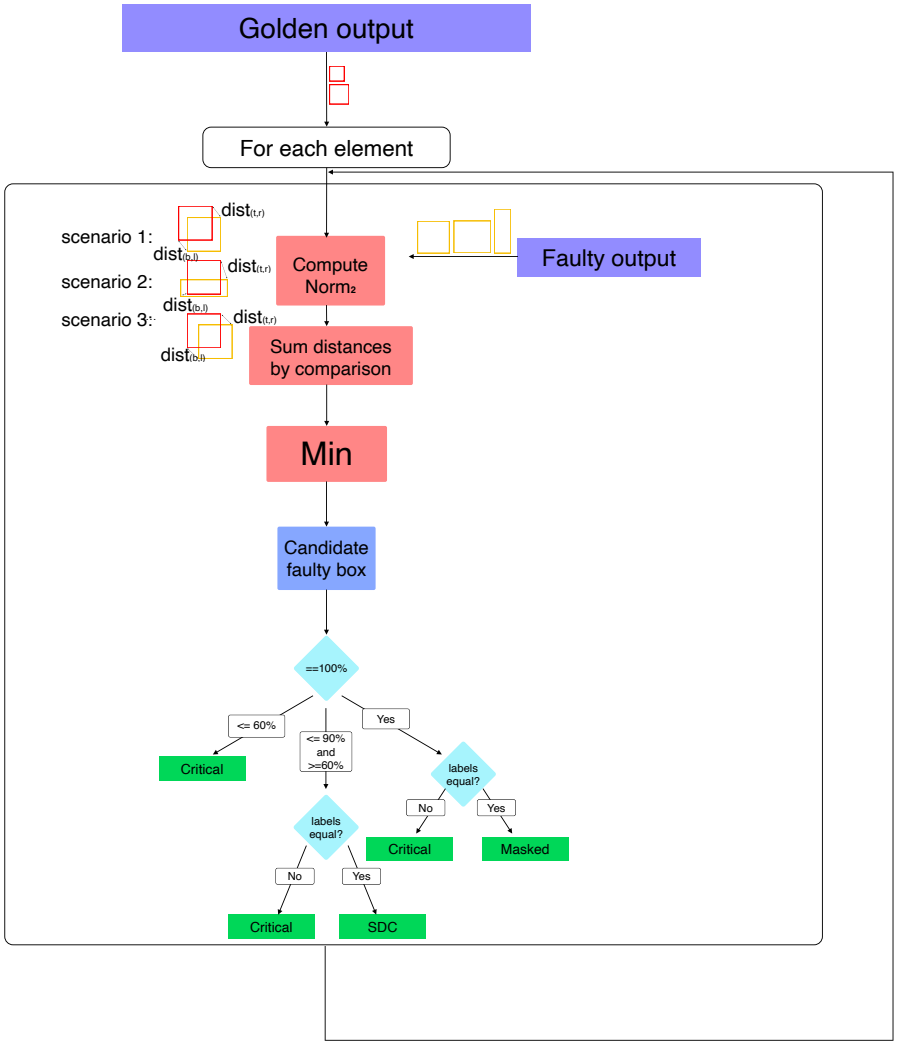


Figure 4.7: Flow chart of the labeling process of the faulty predictions

The IoU score is a metric aimed at the evaluation of an object detection and semantic segmentation algorithm and it computes the ratio between the area of overlap and the union of the bounding boxes representing the ground truth and the prediction.

Under the stated assumptions, the subsequent sections will present the environmental configuration used to train the models for supervised compression in split computing, including the configurations, *loss_criterion* and other hyperparameters, alongside the setup for the FI campaigns i.e. the layer injection type, number of trials, and other hyperparameters that are essential for a comprehensive understanding of the simulation analysis results.

4.3.1 Model: SC Faster RCNN with FPN

Most state-of-the-art object detection models currently used were developed in 2015. They all start with a feature extraction step, followed by the proposal of multiple bounding boxes. They all start with a feature extraction step, followed by the proposal of multiple bounding boxes. These boxes are then filtered based on their prediction confidence level. Differences between these models can be attributed to variations in their box proposal process. The model that was chosen by Matsubara et al. to be implemented in the SC2 framework was Faster RCNN with FPN (Faster Region-based Convolutional Neural Network with Feature Pyramid Network, [40]) which is one of the latest and most optimized versions of RCNN. Input images of varying resolutions are resized, with the longer dimension not exceeding 1000 pixels and the shorter dimension not exceeding 600 pixels. Explanation of technical term abbreviations will be provided upon first usage. Following this resizing, the feature extracted by the *ResNet50* backbone displays consecutive features that correspond to ?? pixels in the input image. For each feature map point (i.e. *Region Proposal*), we initially check for the presence of an object. If an object is found, then we set the coordinates of the Anchor Point to the actual object position.

After performing a 3×3 convolution, two parallel blocks serve as a regressor for the extreme points coordinate of the bounding box and as a classifier for the corresponding label.

Since the backbone employed is identical to the one explained in section 4.2.1, the architecture of the encoder-decoder structure, comprising the separation point appropriate for the student model in the SC2 paradigm, has been injected in the same way as the classification model. This involves the use of identical hyperparameters and configurations during the training process. We decided to keep the same training hyperparameters used by the authors of [1], the only difference in our setup is that, in this case the output of more layers take part to the whole computation of a MSE which computes the distance of the corresponding output of the bottleneck layer and the output of the replaced layer that will be

later added to the MSE computed on the output of the following convolutional blocks.

Eventually, a weights-level FI is carried out with the hyperparameters of tab A.8 and A.8.

4.3.2 Model: SC SSD300

For object detection task we added to the available selection in SC2 another model: SSD300_VGG16, short name for Single Shot multi-box Detector 300 with Visual Geometry Group 16, where:

- *Single Shot*: this means that the tasks of bounding boxes regression and classification are performed in a single forward pass of the network.
- *300*: all the images are previously resized to 300×300
- *MultiBox*: name for the regressor by Szegedy et al., [41]
- *VGG16*: the backbone aimed at feature extraction process.

Following the feature extraction process, the output feature maps are fed into both the multibox regressor and other convolutional layers. The subsequent convolutions aim to generate feature maps with varying aspect ratios and resolutions for new bounding boxes which are consequently fed in the same regressor. The multibox eventually receives 8732 box proposals per class, a number significantly higher than the proposal pool in Faster RCNN with FPN. Our choice in using SSD300_VGG is thus justified by the high precision and speed in bounding boxes generation and classification.

The model trained on COCO2017 is already available in Pytorch, therefore it is ready to be used as teacher model in the context of SC2. Nonetheless, the same process described in section 4.2.2 has been performed in order to fit the bottleneck architecture in the VGG backbone. Under the assumption stated in 4.2.2 the resulting artificial bottleneck architecture is depicted in Fig. 4.8.

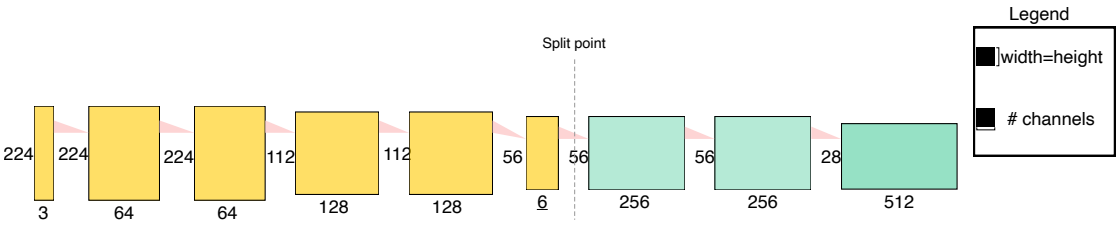


Figure 4.8: SC SSD300_VGG bottleneck architecture.

When performing a convolution with a smaller number of neurons before the split point and then training under the assumptions of SC2, the output feature maps will

try to fit the original data distribution by spreading around the original mean. This procedure is significantly simplified by executing a BatchNormalization once the split point convolution has taken place. Although the authors of [8] have demonstrated through experiments that VGG does not necessitate any normalization step, we have observed that incorporating a BatchNormalization layer considerably facilitates the convergence of the learning algorithm.

In the general outline of the bottleneck layer, a *hardcut* approach is used. This involved redesigning the first 8 convolutional blocks of the VGG16 and adding a bottleneck layer to the 4th block, thereby reducing the number of output channels to match the desired number of layers for the specific configuration being tested.

The FI weights level campaign has been performed with identical hyperparameters as those outlined in table A.9.

4.4 Reliability Evaluation of Semantic segmentation applications

As previously stated in section 2.1.1, semantic segmentation is crucial in computer vision, allowing machines to comprehend visual information with greater detail than traditional object recognition tasks. By assigning a semantic label to each pixel in an image, this method empowers machines not only to identify the presence of objects but also their exact boundaries and spatial relationships within a scene. This capability has been utilized in numerous sectors ranging from autonomous driving and medical imaging to augmented reality and robotics. The requirement for high-precision visual perception continues to increase, and the study and advancement of semantic segmentation algorithms has become paramount in the quest for more sophisticated and capable artificial intelligence systems.

Given its field of applications, assessing the reliability aimed at these tasks becomes fundamental. For what concerns the taxonomy used in order to assign categories *Critical*, *SDC* and *Masked* to the predictions mainly relies on 3 arbitrary thresholds for the global pixel-wise accuracy w.r.t the golden prediction. In particular, the corrupted masks that totally overlap with the golden masks are labeled as *Masked*, when the overlapping is within the range [90%,100%), masks are labeled as *SDC*, otherwise the prediction is *Critical*.

4.4.1 Model: SC Deeplab v3

When forwarding data through the network, the first step is the feature extraction that, in the specific case of SC2, the authors have used ResNet50 such that, the artificial bottleneck injection is performed with the same modalities and the same training setup of the split model shown in section 4.2.1.

Given the computational cost that models for semantic segmentation require for a simple evaluation, we have decided to perform only weight FI.

4.5 Exploration of hardening techniques for SC CNNs

4.5.1 Activation function boundary

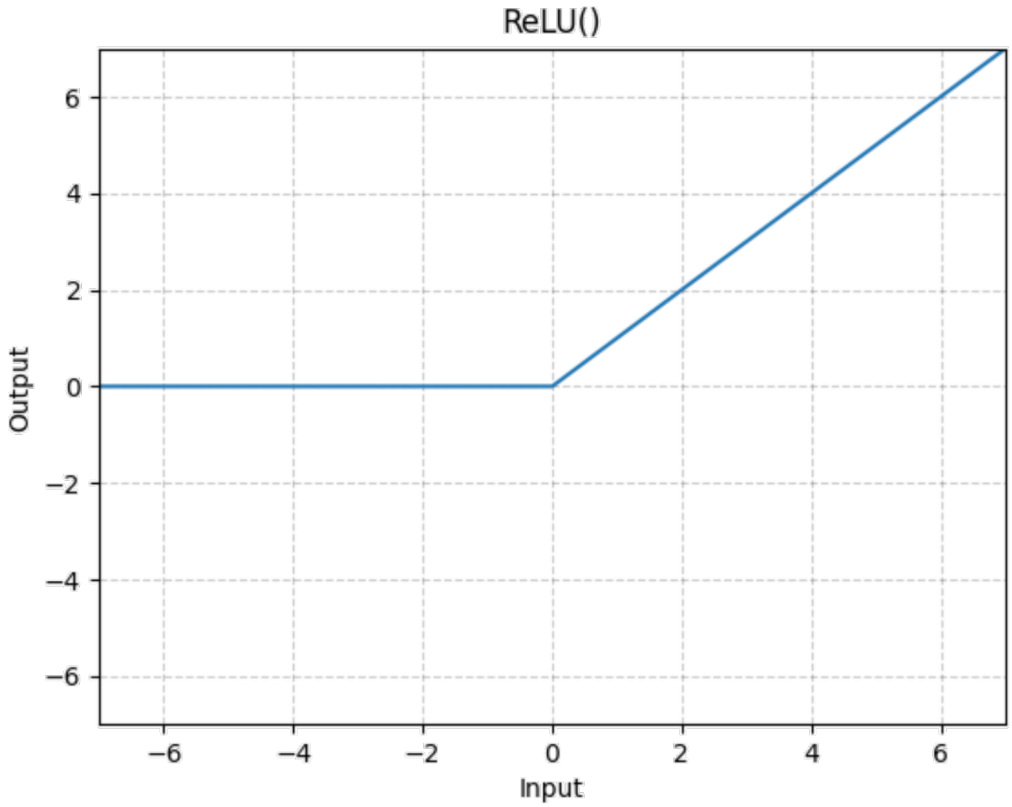


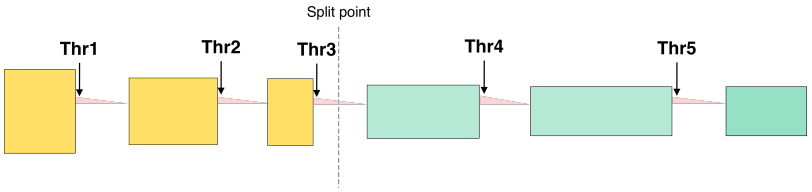
Figure 4.9: ReLU Activation Function

The methodologies of FI as described in sections 4.1.1 and 4.1.2 utilize bit-flips on the MSB, which often lead to tensor cells with values very close to zero, so that they are approximated to 0 due to the machine representation limit, or, in the opposite case, the bit-flips can even lead to very high values. This implies that any scalar product involving the convolutions or BatchNormalization operation may result in NaN, ∞, or extremely high values.

Activation function is one of the components of Convolutional Neural Networks that permits to have control on the single cells of a feature map. Specifically, one of

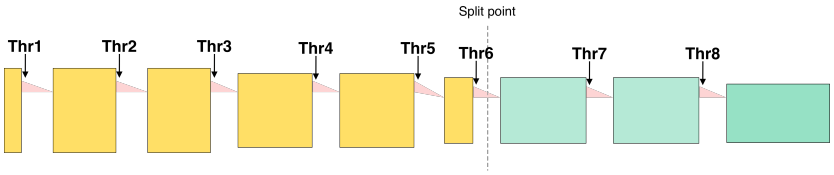
the most commonly used Activation functions in CNNs is *ReLU* which guarantees that the feature maps entries have always positive values applying the function outlined in Fig. 4.9. Since activation functions are always placed at the end of a convolutional block, by using them it is possible to perform a later check after the convolution. ReLU function is the specific case of Hard Hyperbolic Tangent (HardTanH). It is able to clip the values between a specified range and ReLU represents a HardTanH where the range is $[0, +\infty]$. Being able to find suitable Upper Bounds for the outputs of the convolutions could prevent the effect of corruption that would lead values to extremely high values or to $+\infty$. A best practice in machine learning is to normalize the test data with statistics that represent the distribution of the training set, such they could result closer to the training distribution and the model can perform better. So that, an evaluation of the training set have been performed and the maximum per convolutional layer of all the feature maps representing all the training images has been computed. Therefore the computed values have been used to replace the ReLU functions of the original bottleneck layer with HardTanH. Eventually, the newly designed split model is trained in SC2 framework with 6 channels of compression at the split point, and a weight FI have been performed with the same testing hyperparameters of the test performed on the reference student model.

In order to have a general overview of applicability of such hardening technique, it has been adapted to both the bottlenecks of the models implemented during the activities of this work, i.e. splittable MobileNet Classifier (Fig. 4.10) and splittable SSD300_VGG16 (Fig. 4.11).



HardTanh Bounds	Lower	Upper
Thr1	0	20
Thr2	0	10
Thr3	0	20
Thr4	0	18
Thr5	0	17

Figure 4.10: HardTanH Activation Function in splittable MobileNet Classifier



HardTanh Bounds	Lower	Upper
Thr1	0	5
Thr2	0	5
Thr3	0	2
Thr4	0	2
Thr5	0	15
Thr6	0	15
Thr7	0	20
Thr8	0	439

Figure 4.11: HardTanH Activation Function in splittable SSD300_VGG16

4.5.2 Layer swap

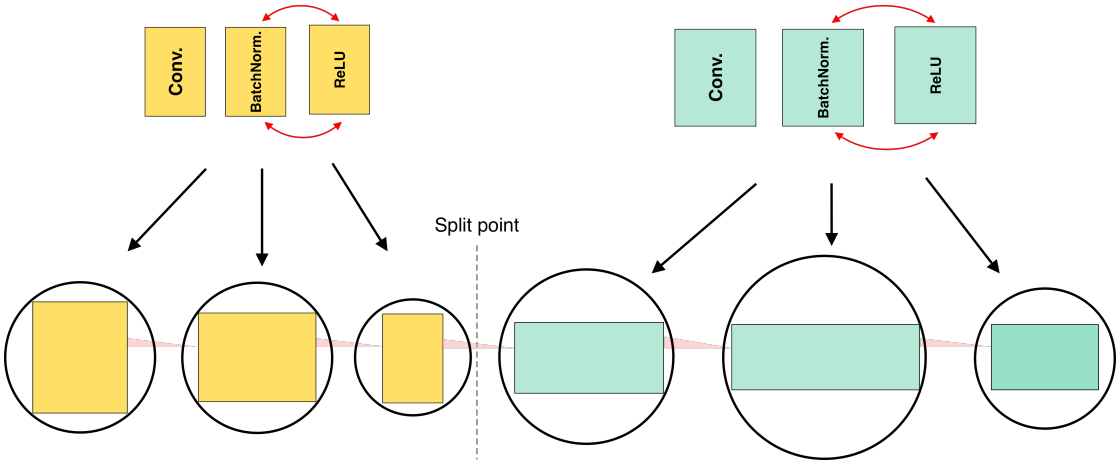


Figure 4.12: Layer Swap in MobileNet Classifier bottleneck

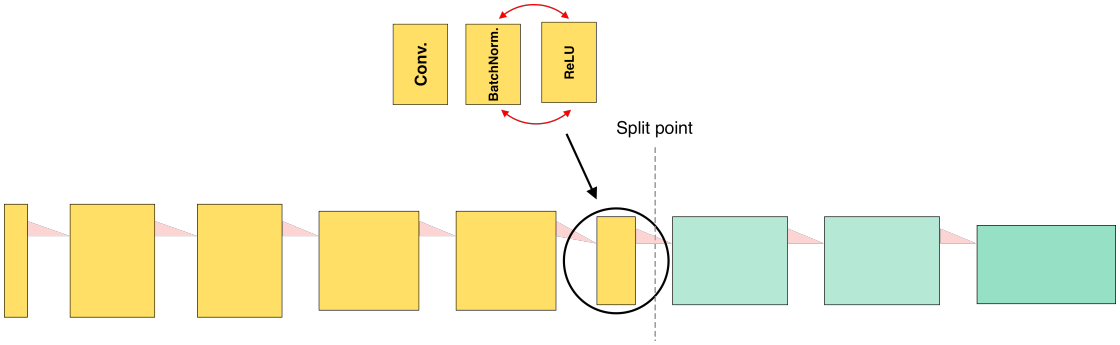


Figure 4.13: Layer Swap in SSD300_VGG16 bottleneck

In section 4.3.2 the importance of the BatchNormalization layer has been highlighted which computes mean and variance statistics within the current batch under analysis and could help the stability of such metrics when the channel reduction is performed in the bottleneck layer. Since the Convolution operation is heavier in terms of required computational cost, it may be more prone to faults than the BatchNormalization layer. Therefore, another hardening technique has been implemented which consists of swapping the BatchNormalization layer, which usually occurs after the convolutional layer, with the ReLU activation function such that it can properly perform the check it is aimed to on the most critical layer. In Fig. 4.12 and 4.13 the visual representation of respectively Split MobileNet and Split SSD300_VGG16, can be found.

4.5.3 Pooling removal

Within a convolutional block, pooling layers are utilized to decrease the spatial dimensions of the feature map without enlarging the parameter space. The design of an encoder-decoder structure like our bottleneck block is commonly guided by the principle of minimizing the reconstruction error, which is eased by increasing the number of learnable parameters, resulting in a more explainable search space. Pooling layers carry out a straightforward function of averaging, taking the maximum, or minimum values. Therefore, by enlarging the kernel sizes of identical convolutional blocks, these layers can be eliminated. The structure of the bottleneck is properly adapted for Split MobileNet and for Split SSD300_VGG16.

4.5.4 Fusion compression

After an ablation study concerning several hardening techniques, it may be interesting to see how their combination performs. A list of the contributions that each hardening proposal can provide to the general structure of the bottleneck is reported.

- *Custom ReLU*: define a proper value range within the expected tensor cells are admitted.
- *Layer swap*: provide a check to the convolution which is the most critical operation.
- *Pooling removal*: decrease the reconstruction error by increasing the size of learnable parameters space

Chapter 5

Experimental Results

This section will provide the results of all the experiments presented in the previous chapter by exploring with the following structure:

- Image Classification task.
- Object Detection task.
- Semantic segmentation task.

Indeed, the statistics about the simulation and training times are presented in the table A.1 where both student and teacher training execution times are evaluated with the best-performing configuration. All the simulations were conducted on

- A workstation HP Z2 G5 with an Intel Core i9-10800 CPU with 20 cores, 32 GB of RAM memory, and equipped with an RTX 3060TI GPU platform including an NVIDIA Ampere architecture with compute capability (CC) 8.6.
- "Legion" cluster was mainly used with 2 Intel Xeon Scalable Processors Gold 6130 2.10 GHz 16 cores, and equipped with 6 NVIDIA Tesla V100 SXM2, 32 GB and 5120 cuda cores (on 6 nodes).

5.1 Image Classification

5.1.1 Model: MobileNet Classifier

The *CIFAR-100* dataset is used as a benchmark in the field of computer vision, playing a crucial role in image classification research and development. With its 100 classes, each containing 600 images of resolution 32×32, it delivers a diverse and challenging set of visual data. Importantly, CIFAR-100 offers a more detailed classification challenge than its previous version, CIFAR-10, covering a

wide range of object categories. This dataset covers a broad range of everyday objects, animals, and natural scenes, requiring advanced techniques in feature extraction and classification. Its rich diversity and complexity make CIFAR-100 an excellent dataset for evaluating the robustness and generalization capabilities of deep learning models in real-world scenarios.

In this context, CIFAR-100 has been used to:

- train and evaluate the teacher model MobileNet.
- train and evaluate its compressed version, i.e. SC MobileNet.
- test the resiliency of SC MobileNet by means of neuron and weights level FI.

Teacher training

The MobileNet teacher model has been trained by trying the most relevant combination of the hyperparameters shown in tab. 5.1. As best practice in training a

Hyperparameter	Value
$start\_leanring\_rate(\gamma)$	0.3,0.15,0.1, 0.05, 0.005, 0.001
$step(s)$	3,5, 10, 15, 30
$weight\_decay\ (\rho)$	$6e^{-5}, 1e^{-5}, 1e^{-4}$
$learning\_rate\_decay(\epsilon)$	0.99, 0.2, 0.15, 0.1, 0.05
$batch\_size$	16, 64, 128, 512, 2048
$momentum$	0.99, 0.89
$warmup\_epochs$	5, 10, 20
num_epochs	100, 200, 300, 400

Table 5.1: MobileNet V3 Small training hyperparameter values.

general-purpose neural network, we started by training the model for 100 epochs, a learning rate (0.1), a multi-step learning rate schedule that decreases the learning rate every 10 epochs of 0.1, and a $weight\_decay = 1e^{-4}$ and we found that the algorithm starts converging too early by reaching the best validation accuracy (30%) after 20 epochs and stop increasing. The fast convergence of the algorithm may be due to a high value of the $leanring\_rate(\gamma)$, then we started exploring lower values but they did not lead to any significant result by reaching a maximum validation accuracy of 40%.

Eventually, by taking as a reference the work of Showlo published in its GitHub repository ([Source]) we decided to use a *Cosinum Annealing Learning Rate scheduler with Warm Up* starting with a learning rate of 0.15 decreased according to the scheduler every 3 epochs of a factor 0.99 and a $weight\_decay(\rho) = 6e^{-5}$. Furthermore the $batch\_size = 2048$, $warmup\_epochs = 5$, $momentum = 0.89$.

As we can see from the learning curves corresponding to loss and accuracy metrics respectively depicted in 5.1 and in 5.2, the trend is smoother than the one described in the other hyperparameter configuration. The loss fast decreases until it is able to reach 2% where it is close to the algorithm convergence and when it is close to the 400° epoch, it reaches its lowest value ($\approx 1.52\%$)

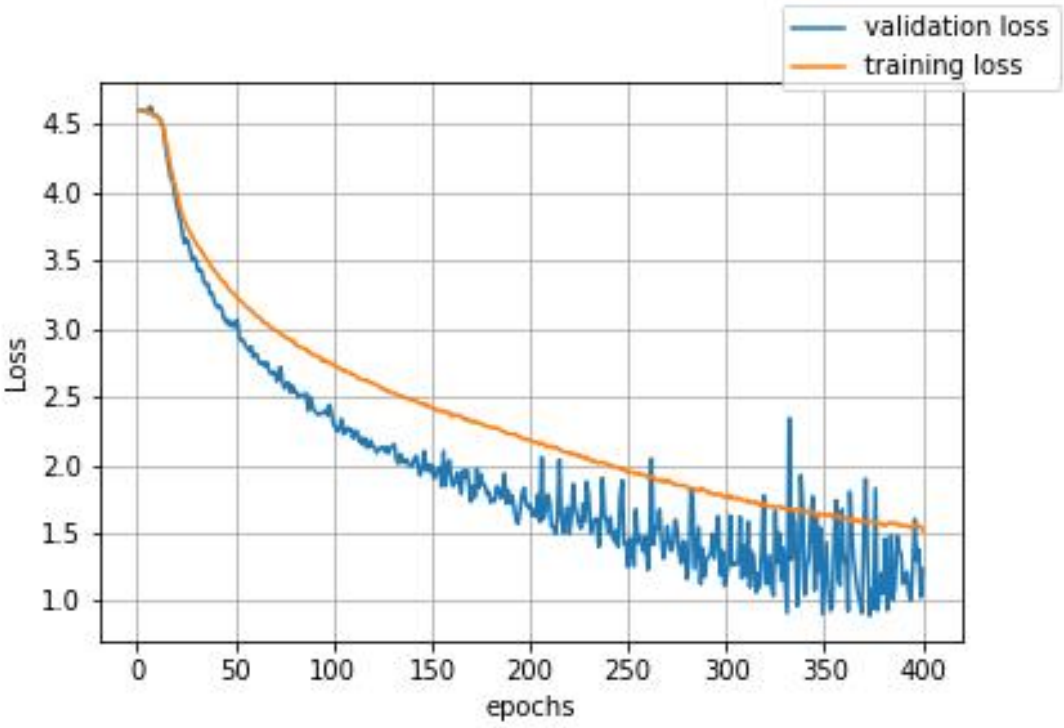


Figure 5.1: MobileNet V3 Small training and validation loss.

The reason behind the final best configuration suggested may be found in the fact that MobileNet is mainly known because of model size optimization which makes the model lightweight due to the limited number of weights and this leads to a search space dimensionality that is more limited than other models like the ResNet50 Classifier. Reducing the search space brings to the avoidance of *curse of dimensionality* phenomenon[1] and consequently the distances between the points delates and the same holds for the hyperplane on which the computed gradient tries to find the optimum.

The table A.3 reports the most relevant configuration with the corresponding

[1]given a fixed volume of a hypercube, as the space dimensionality increases, the distance between 2 edges decreases exponentially where the 2 edges are represented by two general points in the search space. This means that the distance between the points tends to vanish.

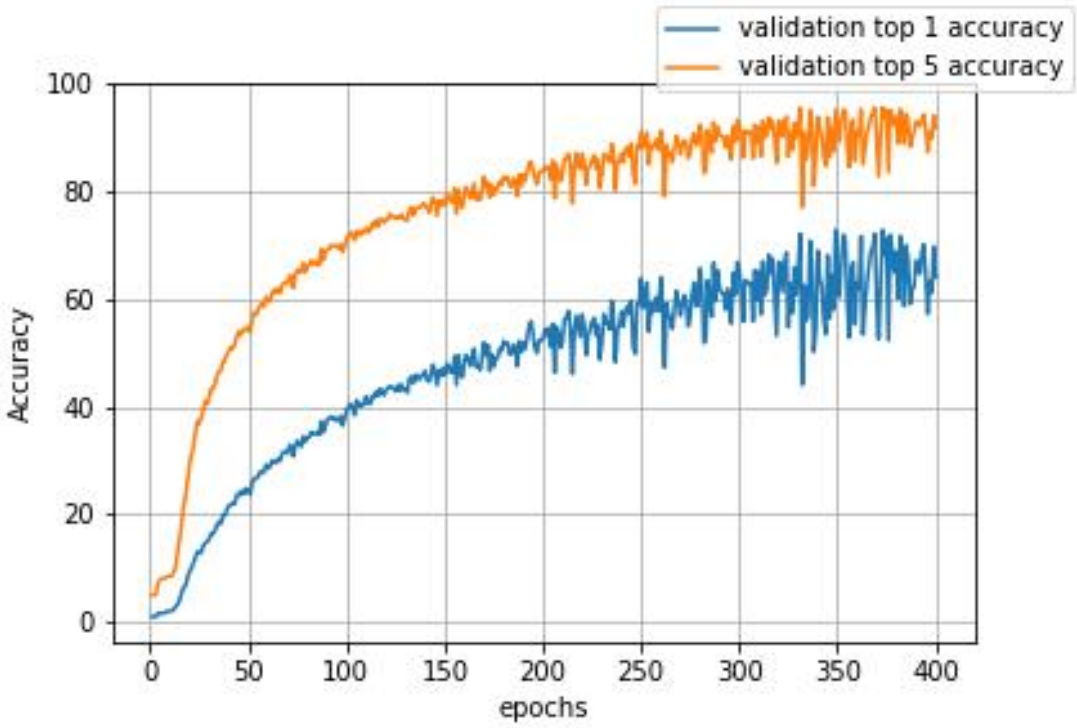


Figure 5.2: MobileNet V3 Small validation Top1 and Top5 accuracy.

best accuracy and loss both at validation and training time.

Once the best configuration has been found and the corresponding model has been trained, MobileNet teacher model for SC2 is available.

Student training

Once the best-performing teacher model has been found, its compressed version and additionally the hardened version with Custom ReLU have been trained.

By means of the strategy presented in section 4.2.2 the *Split MobileNet Classifier* and *Split MobileNet Classifier Custom ReLU* are trained and specifically, we obtained impressive results with a slightly modified version of the original training set up in SC2 framework. Specifically, models are trained for 20 epochs, with $batch\_size = 4$, by freezing the modules that are not replaced by the bottleneck layer, i.e. $layer0$ and the layers that follow the 4^{th}. Furthermore, taking as reference the training pipeline of the teacher a multi-step learning rate schedule has been used with frequent milestones, specifically, they take values from the discrete interval [2,4,6,8,10,12,14,16,18] and consequently, limited learning_rate_step = 0.95 and starting_rate = 0.05. Eventually, the contributions of the MSE loss corresponding to the layers that follow the 4^{th} are summed up and then backpropagated.

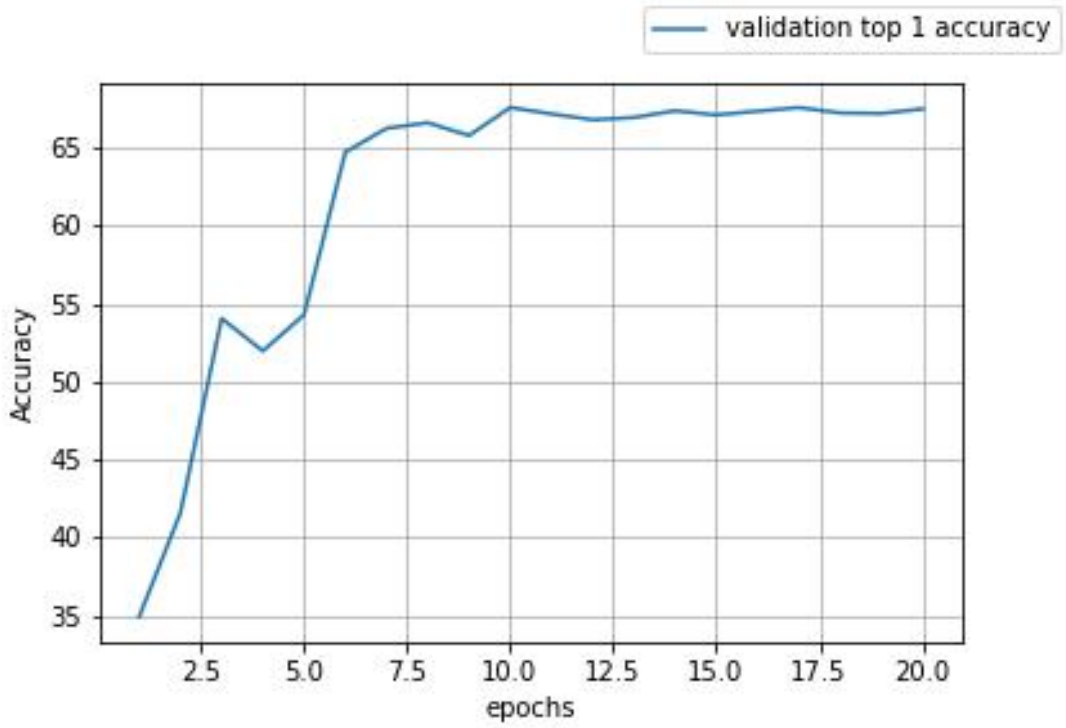


Figure 5.3: SC MobileNet V3 Small Top1 validation accuracy.

Concerning the *Split MobileNet Classifier*, in Figure 5.3, the Top1 validation accuracy see a significant growth during the first 4 epochs, followed by a small drop in performance between the 4^{th} and 5^{th} epochs when the learning rate is 0.045125. However, adjusting the learning rate to 0.04286875 gets it closer to the hyperplane optimum, enabling consistent growth until convergence. The whole process leads to a validation score of Top1, which indicates a performance decrease of merely 5.27% in comparison to the teacher model's performance and the model size results in $10.5MB$ that, compared to the teacher model size, it shows a relative model size decrease of $\approx 20\%$.

Indeed, as Figure 5.4, the Top1 validation accuracy of *Split MobileNet Classifier with custom ReLU* exhibits a rapid increase during the initial 4 epochs, followed by a small plateau between the 4^{th} and 5^{th} epochs when the learning rate is 0.045125. However, when the learning rate is adjusted to 0.04286875, it approaches the hyperplane optimum, leading to continuous growth until convergence. This trend results in a Top1 validation score indicating a performance degradation of only 5.74% compared to the performance of the teacher model.

As stated in section 4.2.2 both weights and neuron-level fault have been injected on the first 4 layers of the architectures and have been carried out focusing only on

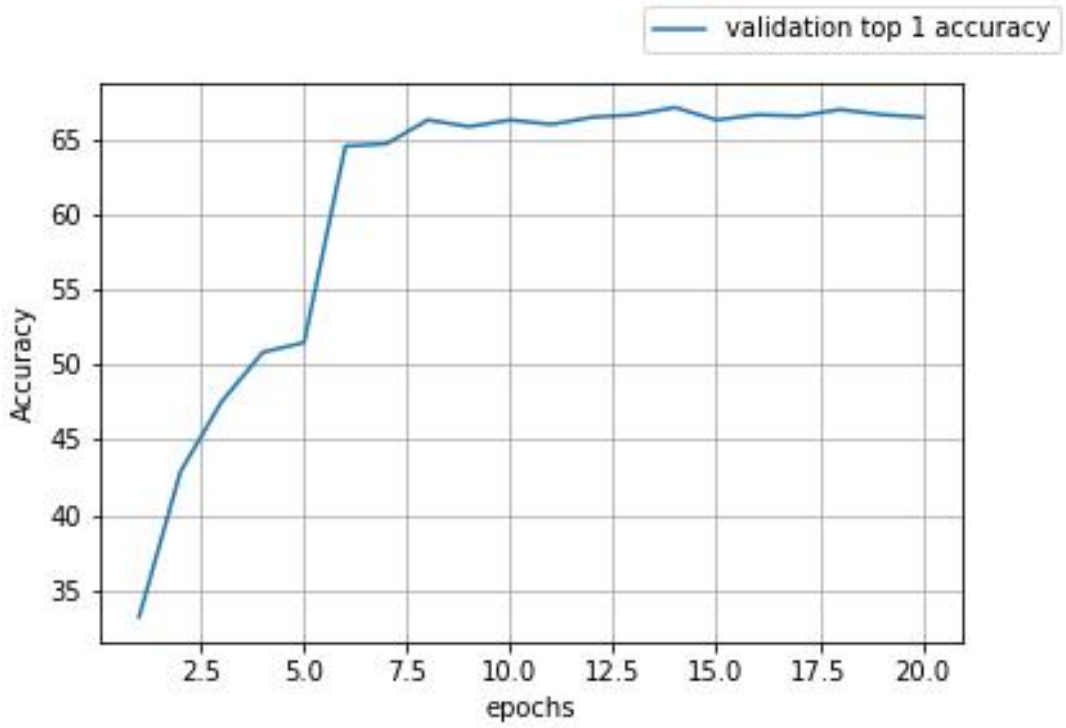


Figure 5.4: SC MobileNet V3 Small with custom ReLU Top1 validation accuracy.

the compression on the split point to 6 channels feature maps.

Weights Fault Injection

The weights FI has been performed with the hyperparameter setting shown in tab 5.2.

Layers	0,1,2,3,4
error margin	$\approx 5\%$
confidence level	99%
probability fault instances	0.5
# of test images	500

Table 5.2: SC MobileNet V3 Small Weight FI hyperparameters.

Fig. 5.5 illustrates, with a log scale, the decline in Accuracy, F1-score, Precision, and Recall metrics concerning the golden performance at various bit faulty positions,

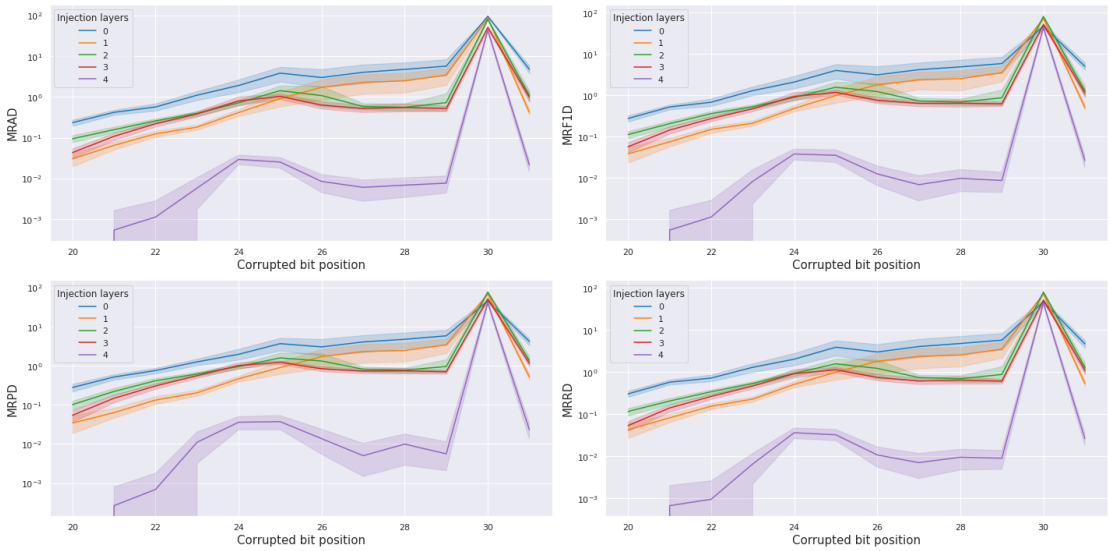


Figure 5.5: SC MobileNet V3 Small Mean Relative Top1 Accuracy, F1-score, Precision and Recall Matrics degradation VS bit faulty position - Weights FI.

according to the following formula:

$$MeanRelativeMetricDegradation = 100 \times \frac{faulty\_metric - golden\_metric}{golden\_metric} \tag{5.1}$$

In the most critical injection, i.e. when the bit-flip occurs on the MSBs, it can be noticed that the more advanced the stage of the network in which the fault occurs, the more resilient the model is. The same trend does not hold for the LSB due to the smaller but not negligible entity of the degradation ($< 1\%$).

Indeed, Fig. A.1 reports the percentage of Top1 Critical, SDC, and Masked predictions for each layer of injection where the Silent Data Corruption[2] percentage, related to the Top1 predictions are $> 80\%$ except for the injection in layer4 when the model seems to have a resilient behavior with the highest masked percentage (33%).

The hardening effects are reported in Fig. 5.6 where the statistics computed especially in the most critical scenario, i.e. 30^{th} bit corruption, show significant improvements when a soft error occurs in *layer*3 and *layer*4. Specifically, tab. A.4 and tab A.5 better outline the comparisons and the improvement in terms of MRAD.

[2]i.e. when prediction accuracy is within 90% and 100% over all the test set images

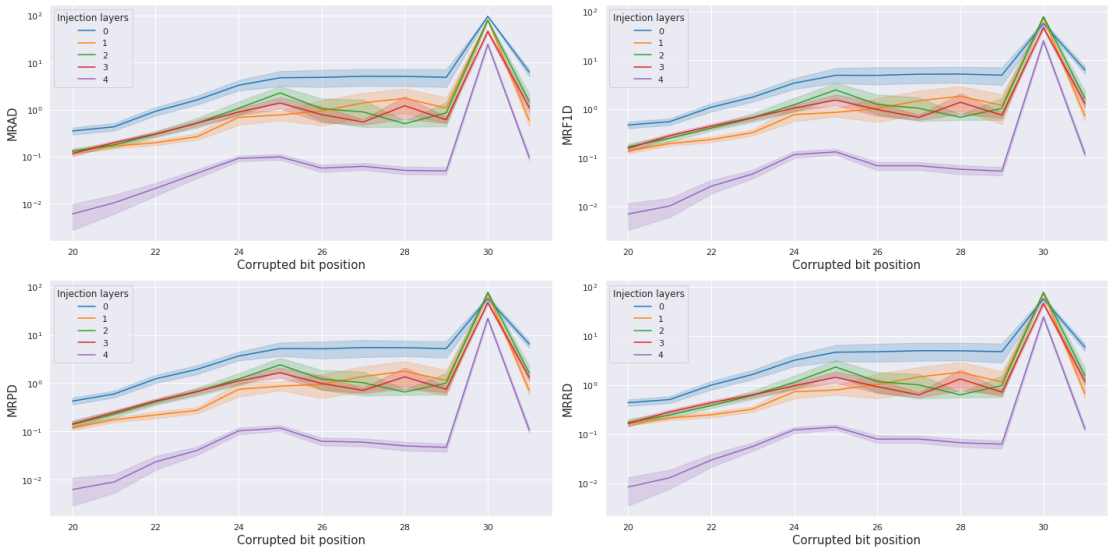


Figure 5.6: SC MobileNet V3 Small with Custom ReLU Mean Relative Top1 Accuracy, F1-score, Precision and Recall Matrics degradation VS bit faulty position - Weights FI.

Neuron Fault Injection

Under the setting shown in Tab. 5.3 neuron FI have been performed.

Hyperparameter	Value
Layers	0,1,2,3,4
Trials	5
tail_size	32×32
block_fault_rate	20%, 40%, 60%, 80%100%
neuron_fault_rate	2%, 4%, 6%, 8%10%
threads	32
# of test images	500

Table 5.3: SC MobileNet V3 Small Neuron FI hyperparameter.

Fig. 5.7 reports the Mean Relative Accuracy Degradation (MRAD) per corrupted bit position showing from bit 30 to bit 25 a metric degradation $> 60\%$ while for the LSB the corrupted model shows a low MRAD ($< 15\%$), specifically the average injection effects can be deduced by tab A.7. The higher MRAD in correspondence of the MSB can be explained by the presence of bottleneck quantization consisting of approximating the $Float32$ resolution to $Int8$ then the considered corrupted bit position causes significant changes in the resulting real number.

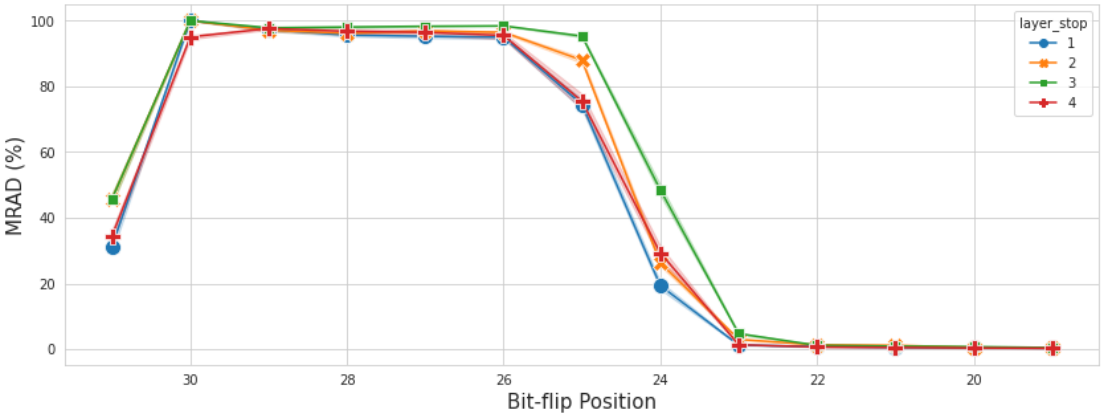


Figure 5.7: SC MobileNet V3 Small Mean Relative Top1 Accuracy Degradation VS bit faulty position by injection layer setting - Neuron FI.

On the other hand, the Custom ReLU implementation shows a MRAD improvement for each injection simulation reported in A.7. Figure 5.8 presents the Mean Relative Accuracy Degradation (MRAD) per corrupted bit position. The results demonstrate a significant degradation in metric performance of more than 40% from bit positions 30 to 25, while a low MRAD value ($< 11\%$) is observed for the LSB corrupted model.

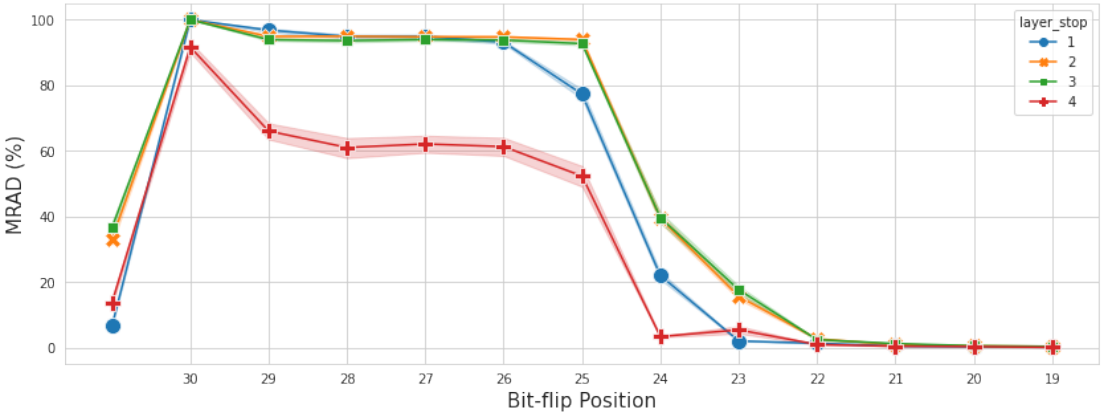


Figure 5.8: SC MobileNet V3 Small with Custom ReLU Mean Relative Top1 Accuracy Degradation VS bit faulty position by injection layer setting - Neuron FI.

5.1.2 SC Model: ResNet50 Classifier

In this context, ILSVRC2012 has been used during fault injection in SC ResNet50. It is one of the most famous Datasets in the field of image classification.

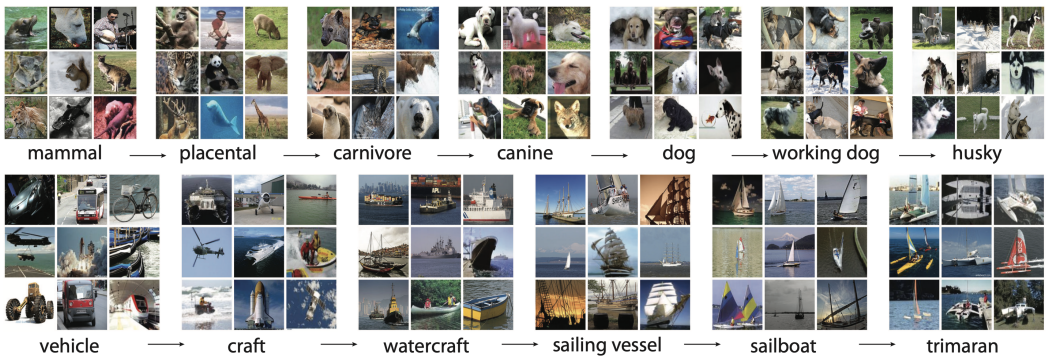


Figure 5.9: Sample from one of the branches of ImageNet. Source: [42]

ImageNet had been continuously updated with new images until the last ILSVRC occurring in 2017 where the team BDAT won the first place with an overall mAP of ≈ 0.73 for the object detection task.

The authors of [1] have decided to use the version of 2012 which contains 10,000,000 labeled images depicting 10,000+ object categories. The competition's validation and test data includes 150,000 photographs from Flickr[3] and other search engines, which have been manually annotated to indicate the presence or absence of 1000 object categories. The categories represent both internal nodes and leaf nodes of ImageNet and do not overlap. A subset of 50,000 labeled images were provided as validation data in the development kit, alongside a list of the 1000 categories. The remaining images, without labels, were used for evaluation purposes.

As stated in section 4.2.1 both weights and neuron-level faults have been injected on the first 4 layers of the architecture over all the possible configurations of channel compression available in SC2, specifically [1,2,3,6,9,12] are the channel depth of the bottleneck feature maps.

Weights Fault Injection

The ImageNet test set is used in the Weights FI campaign with comparable hyperparameter settings as presented in table A.8, in particular, due to the reduced

[3]Flickr is an image hosting and video hosting service, as well as an online community, founded in Canada and headquartered in the United States.

image processing rate (from $\approx 140\frac{img}{s}$ to $\approx 50\frac{img}{s}$), the number of test images must be adjusted accordingly (from 5000 to 500 test images).

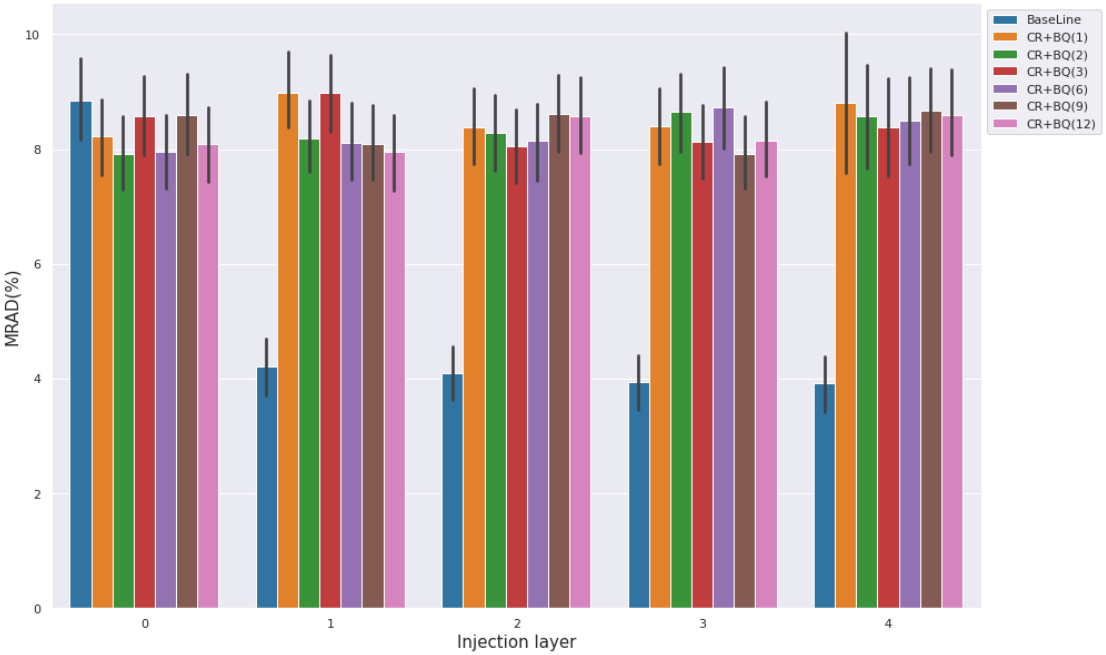


Figure 5.10: SC ResNet50 Mean Relative Top1 Accuracy Degradation VS Injection layer by split configuration (CR+BQ(*)) - Weights FI.

In the context of the Weight FI campaign, SC ResNet50 demonstrates resilience to soft errors, as evidenced by the MRAD in Figure 5.10 never exceeding 10%, except in the case of CR+BQ(1) when injection occurs in layer 4. This outcome is rather curious as one would expect that the earlier the injection, the higher the probability that such a soft error can propagate to the end of the forward pass, thus degrading the performance of the model.

Neuron Fault Injection

For the same reasons mentioned in the previous section, the hyperparameter setting can be described by the following table A.9.

As shown in Figure 5.11, there is a significant decrease in performance for all split configurations when the Neuron FI is performed. This is particularly evident with 12 and 9 channel compression, resulting in an MRAD of approximately 100%. However, the teacher ResNet50 model does not exhibit the same degradation as it possesses a larger number of neurons that can recover possible soft faults in the feature map. Having a lower number of neurons increases the likelihood of

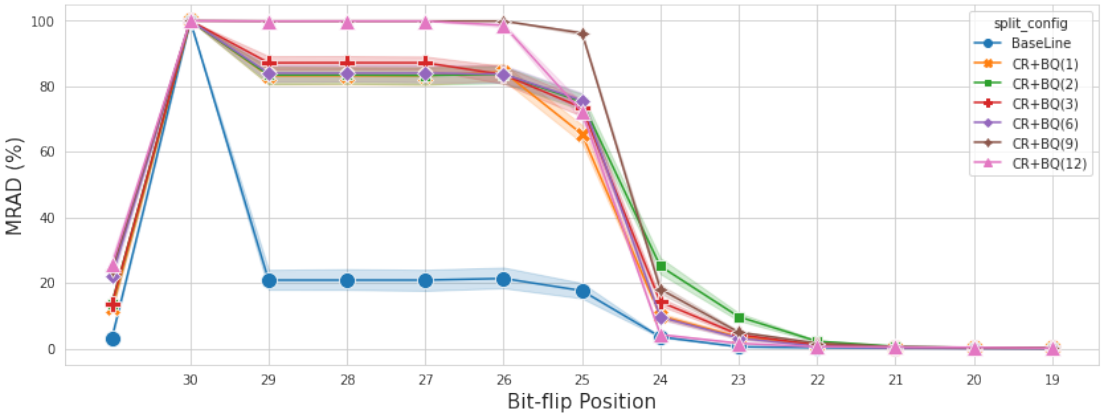


Figure 5.11: SC ResNet50 Mean Relative Top1 Accuracy Degradation VS bit faulty position by split configuration - Weights FI.

degraded attention performance over all the channels of the resulting feature map. Furthermore, bit-quantization can lead to performance degradation due to the reduction of feature map resolutions from $Float32$ to $Int8$, which also makes the previously defined LSB more susceptible to apparently lower effective bit-flips.

5.2 Object detection

5.2.1 Dataset: Coco dataset 2017

The Common Objects in Context 2017 dataset (COCO 2017) presents an extensive and diverse collection of annotated images with different resolutions. It comprises over 200,000 images that serve as a rich visual annotated collection of objects, boxes and caption annotations. COCO 2017 stands out for its ability to present complex, contextual environments, showcasing a wide variety of object categories in real-world scenarios. It categorizes objects and defines stuff categories like sky, water and grass. The dataset offers a richer semantic context, as every image comes with multiple human-generated captions. This contextual granularity presents a challenge for algorithms to comprehensively understand scenes, highlighting the significance of COCO 2017 as a vital resource for object detection and semantic segmentation tasks.

In this context, COCO2017 has been used to:

- train and evaluate its compressed version of pre-trained object detection model, i.e. SC SSD300_VGG16.
- test the resiliency of SC SSD300_VGG16 by means of weights level FI.

- test the resiliency of SC Faster RCNN with FPN by means of neuron and weights level FI.

5.2.2 Student training

Using the approach outlined in section 4.3.2, we trained the *Split SSD300_VGG16* and achieved impressive results with the default configuration set in the SC2 framework reaching an average test IoU score of 45.14% and the model size results $184.4MB$ that, compared to the teacher model size, it, unfortunately, shows a relative model size increase of $\approx 30\%$. Specifically, we trained the model for a total of 20 epochs, with $batch\_size = 16$, by freezing the modules that are not replaced by the bottleneck layer, i.e. the last 2 convolutional blocks of the backbone. Furthermore, following Yoshitomo et al.'s proposed training pipeline, a multi-step learning rate schedule was employed with two milestones. These milestones were selected from the discrete interval of [5,15], and consequently, a high decreasing factor was applied at each milestone (0.1) and 0.001 was used as $start\_\gamma$. The contributions of the Mean Squared Error (MSE) loss correspond to the last two convolutional blocks.

Further training has been performed on the model utilizing the "Custom ReLU" hardening technique, whereby ReLU activation functions have been substituted with HardTanH activation functions. This replacement is executed by correctly setting thresholds, referring to the Vanilla Split SSD300_VGG16. The SC2 training pipeline was utilized with the same hyperparameter configuration outlined in section 5.1.1. The obtained score of $test\_IoU = 41.11\%$ may preliminarly suggest a potential decrease in fault resilience.

Configuration/Layer	score
SC SSD300_VGG16	45.14%
SC SSD300_VGG16 with Custom ReLU	41.11%

Table 5.4: SC SSD300_VGG16 average IoU for split configuration CR+BQ(6).

5.2.3 Fault injection resiliency

As stated in section 4.3.2 only weights-level faults have been injected on the first 4 layers of the architecture and the simulations have been carried out focusing only on the compression on the split point to 6 channels feature maps. Since the object detection consists of 2 downstream tasks (regression of box edges and box label classification), it results harder than image classification task to be performed, then more prone to performance degradation when the FI is run.

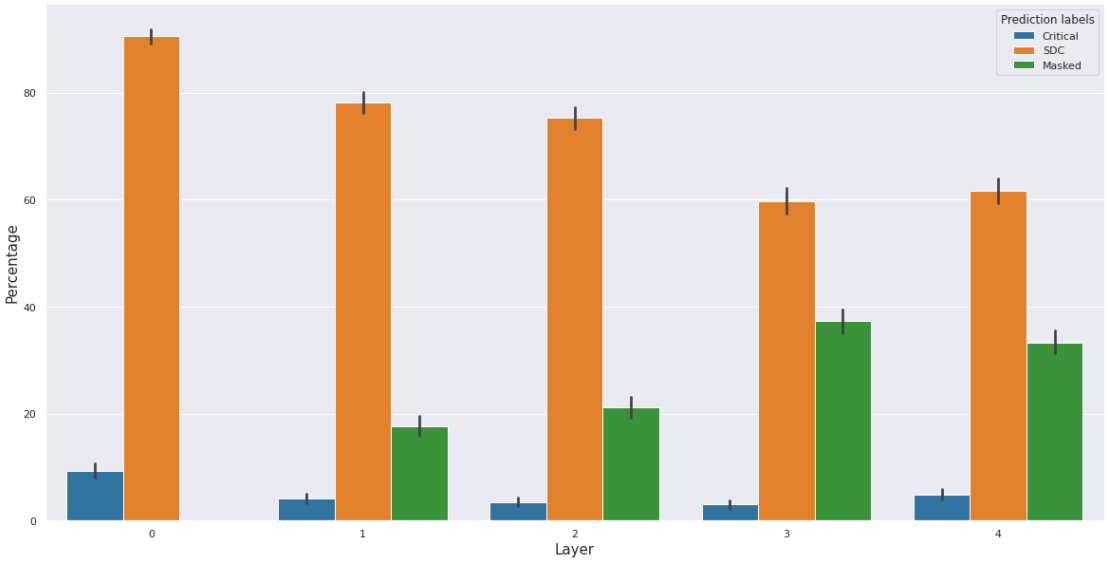


Figure 5.12: SC SSD300_VGG16 Critical, SDC and Masked boxes per layer injection - Weights FI.

Assuming that the number of boxes predicted on the same image, from different models, can change in Fig. 5.12 are reported the percentages of Critical, SDC and Masked boxes over all the boxes predicted on the same image and over all the images. A decreasing trend on the percentage of SDC boxes is noticeable from layer injection 0 to layer injection 4 while the percentage of Masked boxes increases. This confirms the improvement of resiliency performance noticed in section 5.1.1 for image classification.

A deeper analysis focused on statistics evaluated on SDC and on Masked predictions has brought to the results depicted in A.2 and in A.3 where the former explains the tendency of the corrupted model to predict bounding boxes smaller than those predicted by the golden model or in some cases (when MSB are targeted) to not predict anything. Given the aforementioned assumption, in A.3 it is evaluated the number of times that the corrupted model predicts more, less, an equal number of boxes or when it does not predict anything. According to the results regarding the probability distribution of *area ratio* statistic, it is more frequent that the corrupted prediction tends to vanish (in fact, regardless of the injection layer, tab. A.10 reports the absolute percentages of all the cases).

For this task, the metric degradation is determined by the complement of the IoU score that assesses the percentage of non-overlapping regions between the corrupted and golden predictions. The obtained outcomes are presented in tab. 5.5.

Let us now focus on how the SSD300_VGG16 performs when the Custom

Split Configuration/Layer	0	1	2	3	4
CR+BQ(6)	42.75%	39.55%	39.19%	38.85%	39.73%

Table 5.5: SC SSD300_VGG16 IoU degradation - Weights FI.

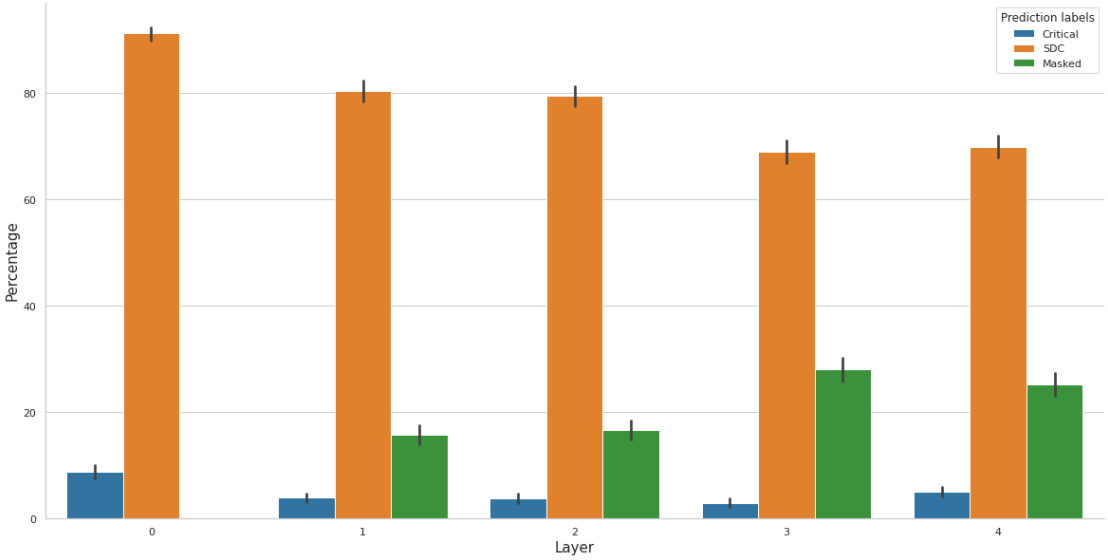


Figure 5.13: SC SSD300_VGG16 with Custom ReLU Critical, SDC and Masked boxes per layer injection - Weights FI.

ReLU hardening technique (section 4.5.1) is employed. If plots in Fig. 5.13 and in Fig. 5.12 are compared, it is noticeable that such a hardening technique is not as effective as it was proven for the image classification task. In fact, we can deduce that the percentages of Critical and SDC prediction increase for this model due to the performance degradation w.r.t. the vanilla model. This happens despite in Fig. A.5, there are no more cases in which the model does not predict anything but, on the other hand, the above-mentioned phenomenon of vanishing boxes becomes more predominant as it can be noticed also in A.4.

Therefore the resulting IoU degradation is resumed in tab. 5.6

5.2.4 SC Model: Faster RCNN with FPN

In SC2, the pretraining of Faster RCNN with FPN was already provided by Yoshitomo et al., therefore it was possible to perform both weight and neuron FIs with all the provided configuration (i.e., $[CR+BQ(1), CR+BQ(2), CR+BQ(3), CR+BQ(6), CR+BQ(9), CR+BQ(12), BaseLine]$ where baseline corresponds to the teacher model).

Split Configuration/Layer	0	1	2	3	4
CR+BQ(6)	43.34%	41.21%	41.21%	40.54%	41.71%

Table 5.6: SC SSD300_VGG16 with Custom ReLU IoU degradation - Weights FI.

Weights Fault Injection

The table 5.7 shows the hyperparameter setting of the FI campaign performed on Faster RCNN with FPN

Layers	0,1,2,3,4
error margin	$\approx 5\%$
confidence level	98%
probability fault instances	0.5
# of test images	100

Table 5.7: SC Faster RCNN with FPN weights fault injection hyperparameters

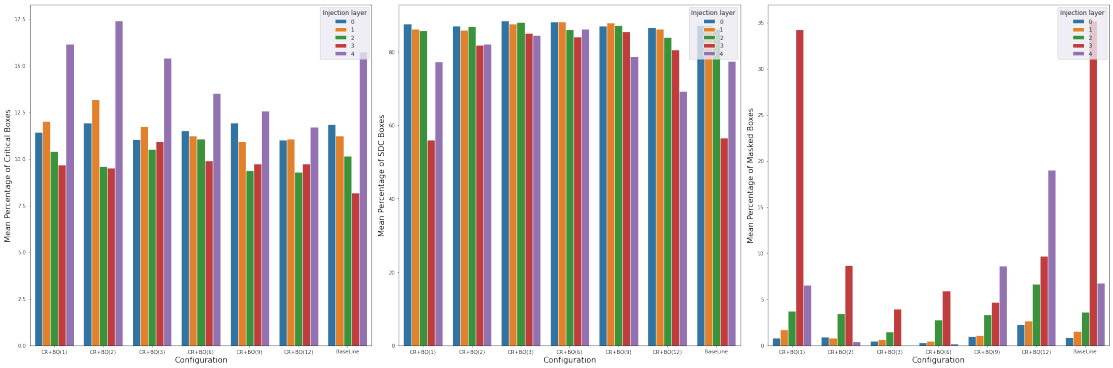


Figure 5.14: SC Faster RCNN with FPN Critical, SDC and Masked boxes per layer injection - Weights FI.

From a preliminary assessment (Fig. 5.14), all configurations present a prevalence of SDC predictions (around 80%) nevertheless, when a fault occurs in layer 3 for channel reduction to 1 and for the teacher model the architectures seem to be more fault resilient with a Masked prediction percentage of $\approx 35\%$.

Further analysis have been carried out in order to explain such general behavior focusing on SDC and Critical predictions. For these cases, the distribution of the area ratio results (A.6) shows a slight trend for this model to boxes vanishing. Such trend is then confirmed in Fig. A.7 which depicts a striking prevalence of the cases

in which the corrupted model does not predict anything surpassing the 50%.

Eventually the IoU degradation is represented in tab. 5.8.

Configuration/Layer	0	1	2	3	4
CR+BQ(1)	13.95%	14.64%	13.70%	13.80%	16.09%
CR+BQ(2)	12.44%	13.35%	10.86%	11.35%	15.27%
CR+BQ(3)	13.03%	13.56%	13.04%	13.89%	15.64%
CR+BQ(6)	11.25%	11.12%	11.59%	11.07%	12.48%
CR+BQ(9)	11.59%	11.05%	10.08%	10.77%	12.07%
CR+BQ(12)	13.12%	13.58%	12.33%	12.97%	14.02%
BaseLine	14.30%	13.82%	13.44%	12.34%	15.74%

Table 5.8: SC Faster RCNN with FPN IoU degradation - Weights FI.

5.3 Semantic segmentation

5.3.1 Dataset: Pascal VOC 2012

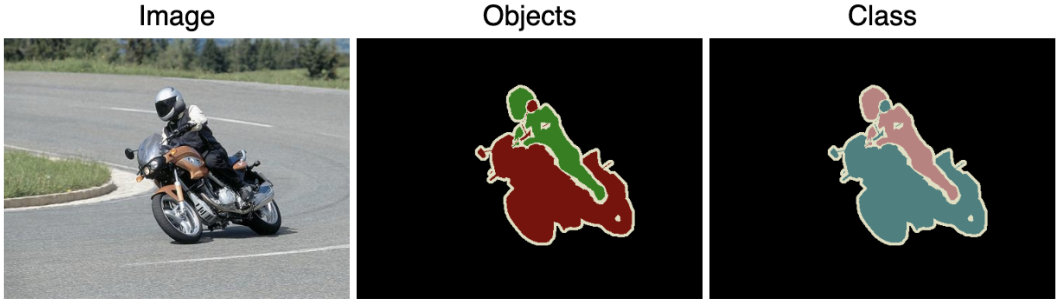


Figure 5.15: *Segmentation*: Generating pixel-wise segmentations giving the class of the object visible at each pixel, or "background" otherwise. [Source]

The Pascal VOC 2012 dataset is a cornerstone in the field of computer vision, particularly renowned for its relevance in object detection and semantic segmentation research. Originally introduced as part of the Visual Object Classes (VOC) challenge, this dataset encompasses a diverse array of images spanning 20 distinct object categories, encompassing everyday objects like cars, pedestrians, and various animals. The dataset is meticulously curated and comprises a training set consisting of approximately 1,464 images, a validation set with roughly 1,449 images, and a test set incorporating around 1,456 images. Each image is meticulously annotated, providing not only object-level labels but also precise bounding box coordinates for

Figure 5.16: Action Classification: Predicting the action(s) being performed by a person in a still image. [Source]

accurate object localization, as well as pixel-level segmentation masks for detailed semantic understanding.

In this context, the testing of the DeeplabV3 model's robustness was conducted using Pascal VOC 2012.

5.3.2 Fault injection resiliency analysis

Due to the limitations of the resources at our disposal, weights FI with all configurations have been performed with the following hyperparameter setting: The

Layers	0,1,2,3,4
error margin	$\approx 5\%$
confidence level	98%
probability fault instances	0.5
# of test images	50

Table 5.9: SC DeepLabV3 weights fault injection hyperparameters.

obtained results are depicted in fig. 5.17 that mainly highlights the high sensibility to soft faults of SC DeepLabV3. In fact it is noticeable that models trained in all split configurations (CR+BQ) present a Critical prediction percentage even higher than the previous Neuron FI which demonstrated to be the most effective reaching exceeding the 30% in the worst performing split configurations, i.e. CR+BQ(3), CR+BQ(6), CR+BQ(9), CR+BQ(12).

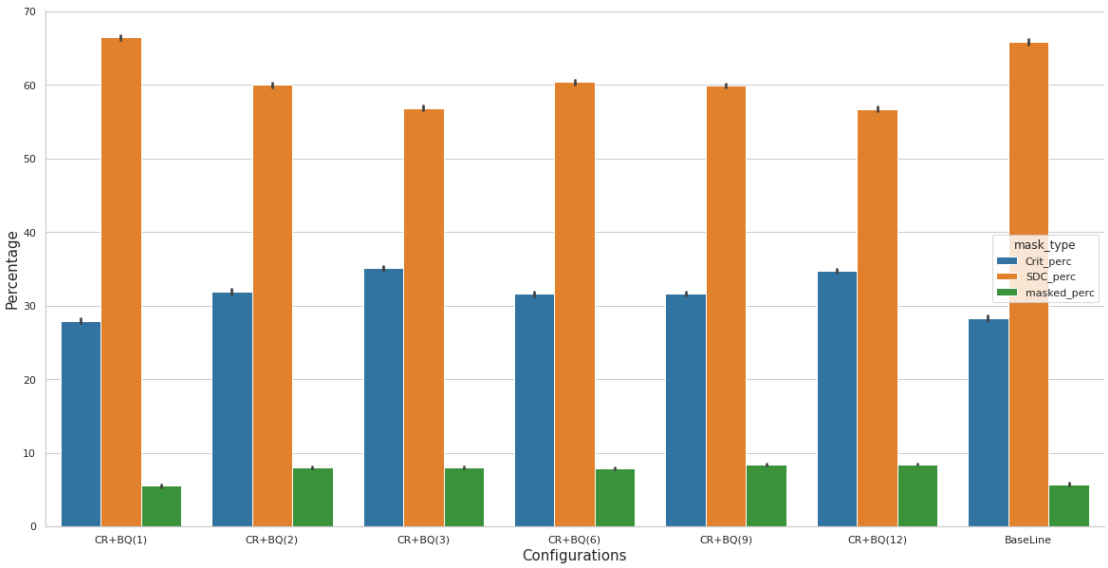


Figure 5.17: SC DeepLabV3 prediction percentages - Weigths FI.

Chapter 6

Conclusions and future works

In conclusion, this dissertation builds on the state-of-the-art of Split Computing (Supervised Compression for Split Computing from Matsubara et al. [1]), which distributes DNNs between edge devices and the cloud within IoT systems. In recognition of the constraints caused by limited computing resources, the concept of SC is investigated as a means to deploy DNNs efficiently on IoT devices. To tackle the critical problem of system reliability, specifically in safety-critical situations, a FI system is developed to replicate software effects due to hardware faults such that it is possible to study their effect on the reliability of SC. The integration of software hardening methods was informed by the findings from these simulations, improving the robustness of the delicate deep neural network models. Our contributions also include extending the collection of models for SC, training them using Supervised Compression, and investigating the most suitable hardening technique. Looking ahead, the dissertation emphasizes the necessity for a standardized benchmark suite to evaluate model resilience in AI applications on IoT systems.

From the experimentation conducted on the initial SC models, it became apparent that the model occasionally failed to make predictions due to the occurrence of NaN values generated by the bit flip corresponding to the MSB, which caused the convolution to produce values tending towards ∞ or $-\infty$. Therefore, it was expected that the most significant improvement would come from the hardening technique involving the upper bound of the ReLU activation function, which was confirmed by the FI campaign in the SC image classification model.

Therefore, it is significant to highlight the findings related to the implementation of a custom ReLU hardening technique on the MobileNet V3 Small model. The research effectively demonstrated how the adapted MobileNet V3 Small, integrated with the custom ReLU approach, exhibited a noteworthy enhancement in fault

tolerance. This was particularly evident under conditions of weights-level FI, where the model's resilience to such perturbations was tested. The CIFAR-100 dataset served as a challenging benchmark, allowing for a comprehensive assessment of the model's performance under various fault scenarios.

Precisely, the custom ReLU implementation mitigated the impact of faults injected at both the neuron and weight levels, by, on average, decreasing the MRAD of 6.278. When compared to the standard MobileNetV3Small model, the custom ReLU variant displayed a more robust behavior, particularly in scenarios involving the most critical bit-flip errors at higher bit positions. These findings underscore the effectiveness of the custom ReLU technique in enhancing the fault tolerance of MobileNetV3Small, making it a valuable contribution to the field of resilient SC deep learning models, particularly in applications where reliability under fault conditions is paramount. Based on the findings some proposals for future developments are presented:

- Extend the range of models available to provide a comprehensive and reliable assessment of the reliability of SC architectures.
- Despite the good performance of the models trained in the SC context, a fine-tuning step can be performed on different datasets in order to improve their generalization capabilities.
- Examine the resilience of the models hardened with the untested techniques described in section 4.5, consisting of *Pooling removal* and *Layer swap*. Furthermore, since the literature (such as [36]) reoorted significant results for the layer swap technique together with the custom activation function, a noteworthy experiment could explore the fusion of these two strategies.
- As soon as a more extensive and comprehensive benchmark is available and the most sensible parts of the architecture are detected, it would be interesting to explore Fault Aware Training (FAT) as a hardening technique.

Appendix A

Experimental Results

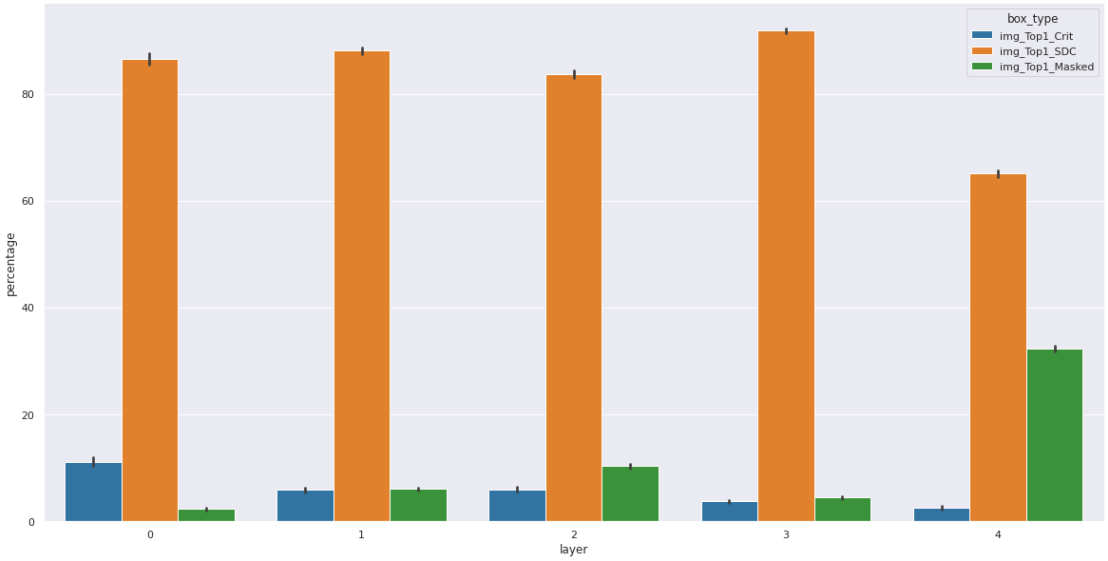


Figure A.1: SC MobileNet V3 Small, SDC and Masked predictions per layer - Weights FI.

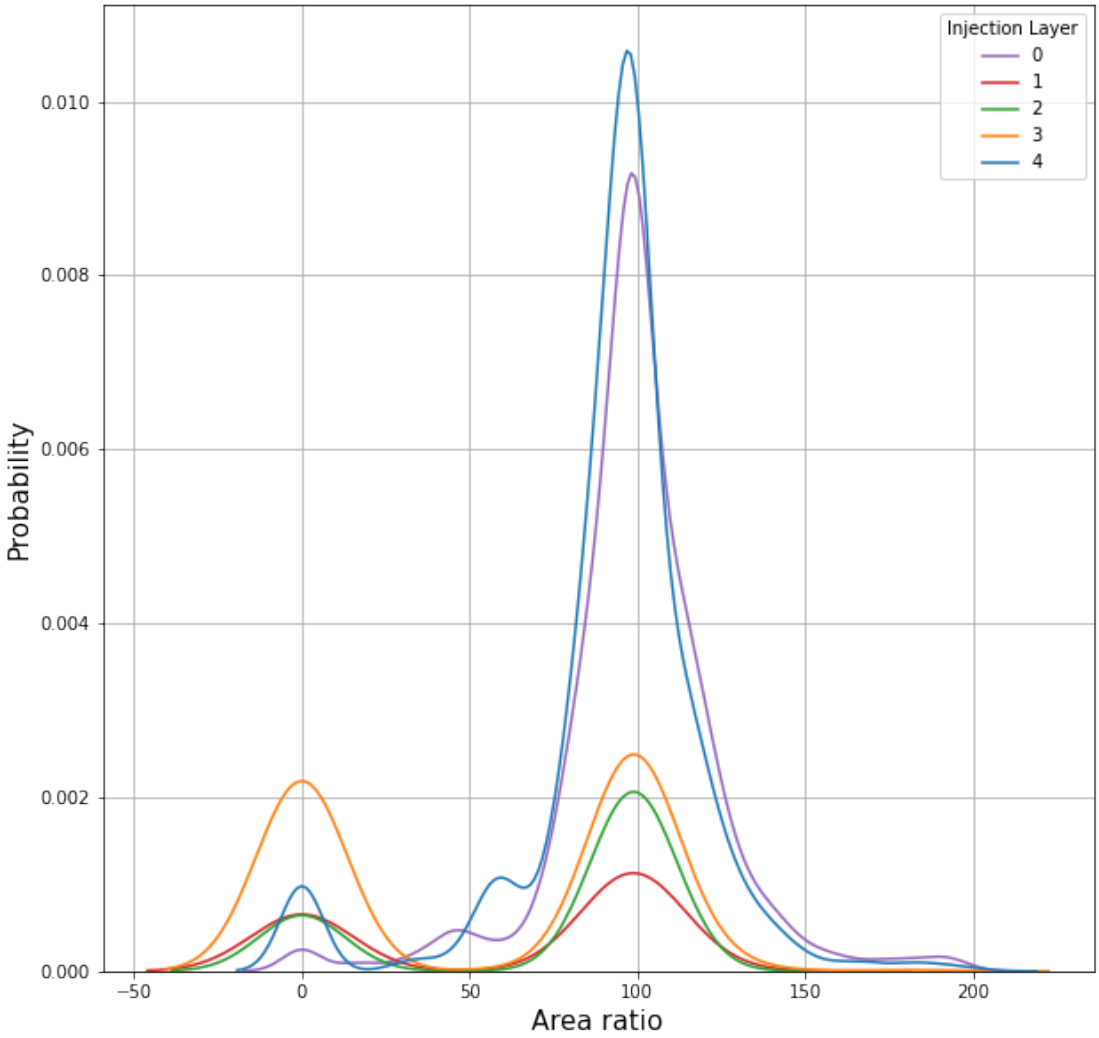


Figure A.2: SC SSD300_VGG16 hard-cut area ratio per layer injection.

		Fault Injection			
	Training	# Corrupted Weights	Weight	# Corrupted Neurons	Neuron
MobileNet V3 Small	$\approx 5h$				
SC MobileNet V3 Small	$\approx 4h$	27058	$\approx 8gg : 8h$	6496	$\approx 4gg$
SC MobileNet V3 Small Custom ReLU	$\approx 4h$	27058	$\approx 8gg : 8h$	6496	$\approx 4gg$
SC ResNet50		54016	$\approx 16gg : 8h$	6496	$\approx 20gg : 12h$
SC SSD300_VGG16	$\approx 10h$	8275	$\approx 8gg : 12h$		
SC SSD300_VGG16 Custom ReLU	$\approx 10h$	8275	$\approx 8gg : 12h$		
SC Faster RCNN with FPN		54016	$\approx 20gg : 20h$		
SC DeepLabV3		54016	$\approx 20gg$		

Table A.1: Execution time of performed experiments. The time represented is evaluated without the use of distributed jobs. However, given the HPC provided by Politecnico di Torino, multiple jobs were initiated and parallelized to achieve feasible simulation times.

start_γ	*step*	*ρ*	*ϵ*	*batch_size*	*momentum*	*num_epochs*
0.15	3	$6e{-}5$	0.99	512	0.89	400
0.1	20	$1e{-}4$	0.1	64	0.99	100
0.1	10	$1e{-}4$	0.1	64	0.99	100
0.1	**3**	**6e-5**	**0.99**	**2048**	**0.89**	**400**
0.1	10	$1e{-}4$	0.05	64	0.99	100
0.3	3	$6e{-}5$	0.99	64	0.89	100
0.2	3	$6e{-}5$	0.99	32	0.89	400

Table A.2: Relevant MobileNet V3 Small hyperparameter configuration trained.

Train			Val		
Top1 Acc. (%)	Top5 Acc. (%)	Loss (%)	Top1 Acc. (%)	Top5 Acc. (%)	Loss (%)
58.79	84.8	1.53	73.37	95.92	0.87
23.14	51.06	3.15	27.54	57.55	2.88
25.30	54.16	3.04	29.96	61.44	2.74
58.55	**84.87**	**1.51**	**74.82**	**95.54**	**0.89**
25.80	54.78	3.01	30.86	61.93	2.71
53.64	81.91	1.73	68.20	93.89	1.07
57.38	84.31	1.58	71.31	92.27	0.83

Table A.3: Models performance trained under the configuration listed in A.2.

Model/Injection layer	0	1	2	3	4
SC Mobilenet	100%	100%	100%	100%	99.69%
SC MobileNet with custom ReLU	100%	100%	100%	99.07%	98.15%

Table A.4: SC Mobilenet MRAD per layer when corrupted_bit = 30 - Weights FI.

Model/Injection layer	0	1	2	3	4
SC Mobilenet	95.92%	86.15%	81.06%	50.90%	43.42%
SC MobileNet with custom ReLU	96.10%	80.94%	79.19%	45.82%	24.30%

Table A.5: SC Mobilenet MRAD per layer - Weights FI.

Model/Injection layer start and layer stop	0-1	1-2	2-3	3-4
SC Mobilenet	100%	99.98%	100%	95.04%
SC MobileNet with custom ReLU	100%	99.97%	99.98%	91.63%

Table A.6: SC Mobilenet MRAD per layer when corrupted_bit = 30 - Neuron FI.

Model/Injection layer start and layer stop	01	12	23	34
SC Mobilenet	47.05%	50.23%	53.08%	47.98%
SC MobileNet with custom ReLU	45.54%	50.24%	50.56%	31.61%

Table A.7: SC Mobilenet MRAD per layer - Neuron FI.

Layers	0,1,2,3,4
error margin	$\approx 5\%$
confidence level	99%
probability fault instances	0.5
# of test images	5000

Table A.8: SC ResNet50 Weight FI hyperparameters - Weights FI.

Hyperparameter	Value
Layers	0,1,2,3,4
Trials	5
tail_size	32×32
block_fault_rate	$20\%, 40\%, 60\%, 80\%100\%$
neuron_fault_rate	$2\%, 4\%, 6\%, 8\%10\%$
threads	32
# of test images	5000

Table A.9: SC ResNet50 Weight FI hyperparameters - Neuron FI.

Majority	percentage (%)
F_majority	15.87
G_majority	35.39
G_F_equality	18.03
F_not_predicted	30.70

Table A.10: SC SSD300_VGG16 prediction majority - Weights FI. It represents the number of times (in percentage) that the #Predictions from corrupted SC SSD300_VGG16 is $>, <, =$ to #Predictions from fault free SC SSD300_VGG16 when considering only SDC and Critical predictions.

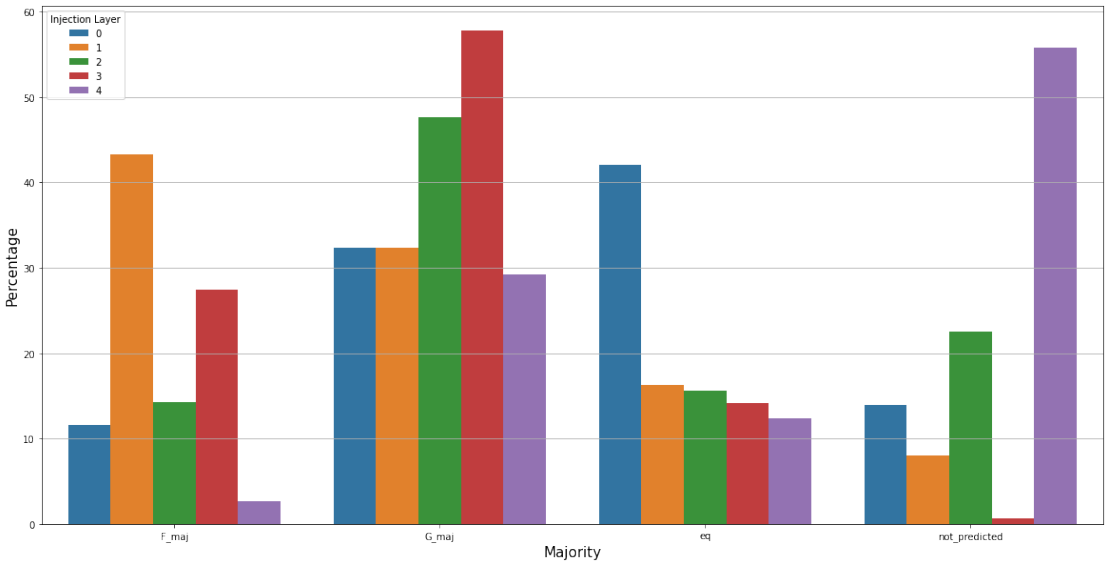


Figure A.3: SC SSD300_VGG16 hard-cut majority of prediction per layer injection. The percentage of cases in which the number of boxes predicted by the faulty SC SSD300_VGG16 is smaller (F_maj), larger (G_maj) or equal (eq) than the number of boxes predicted by the corresponding golden model, or when the corrupted model made no predictions (not_predicted).

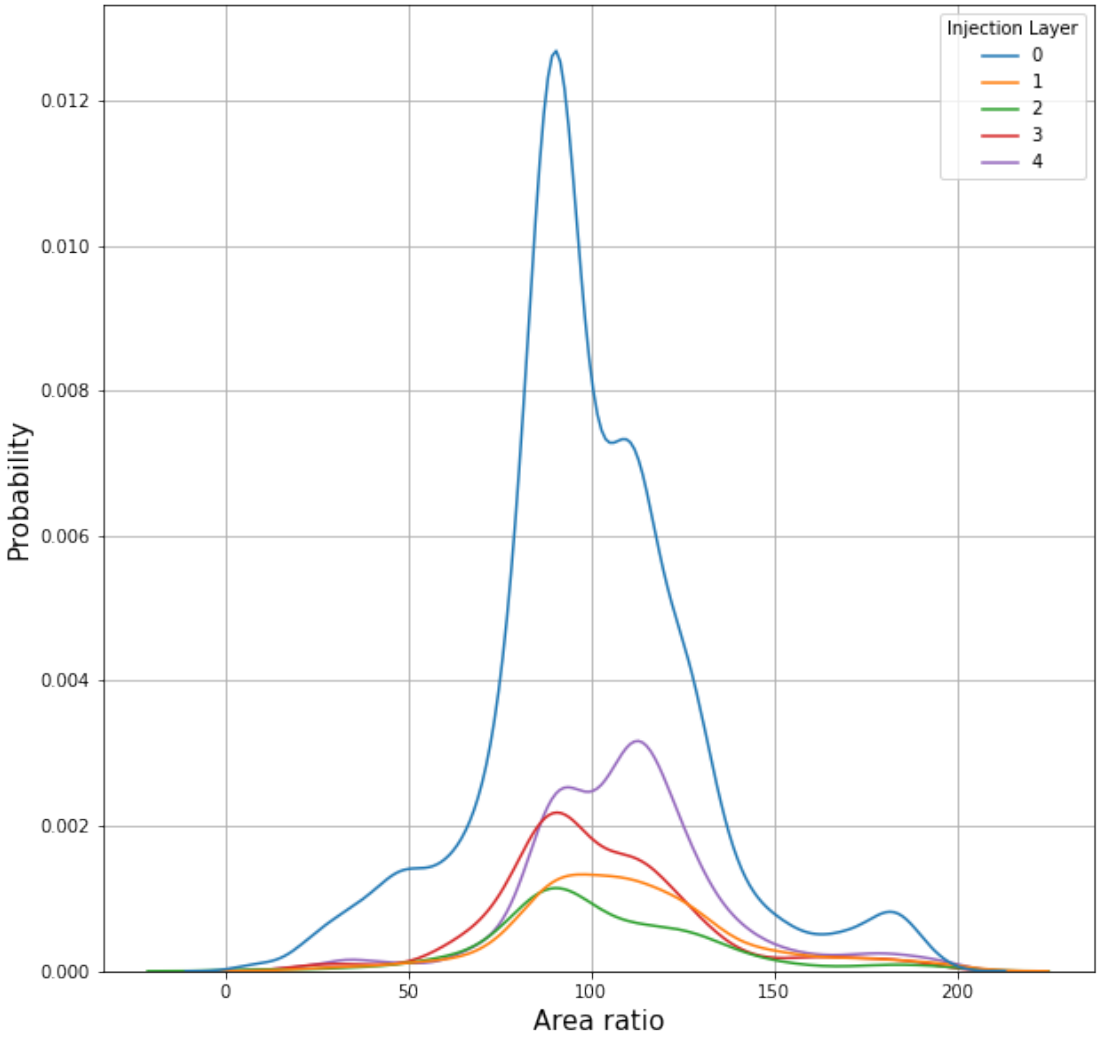


Figure A.4: SC SSD300_VGG16 Custom ReLU area ratio per layer injection.

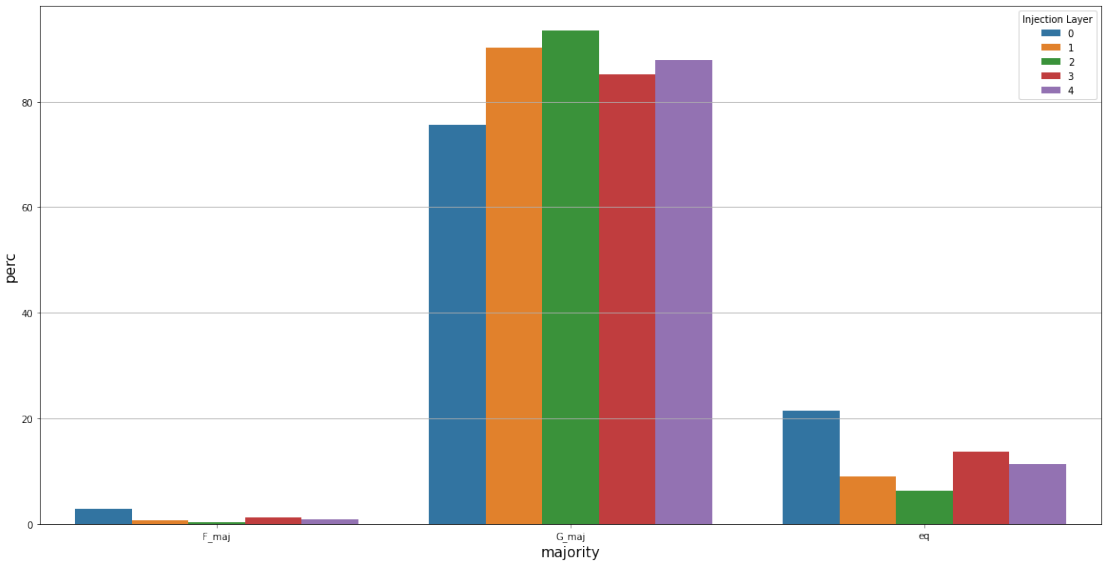


Figure A.5: SC SSD300_VGG16 Custom ReLU majority of prediction per layer injection. The percentage of cases in which the number of boxes predicted by the faulty SC SSD300_VGG16 Custom ReLU is smaller (F_maj), larger (G_maj) or equal (eq) than the number of boxes predicted by the corresponding golden model, or when the corrupted model made no predictions (not_predicted).

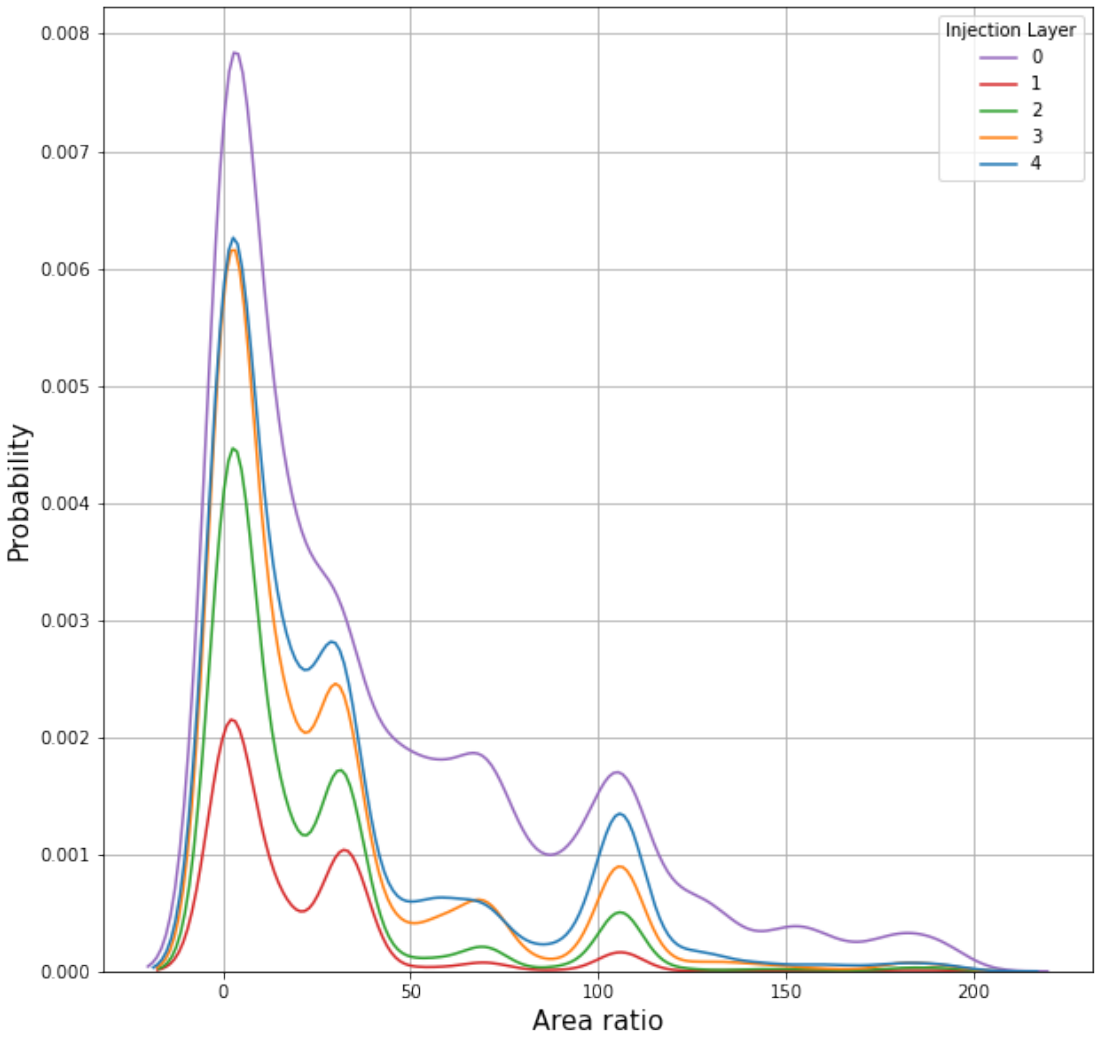


Figure A.6: SC Faster RCNN with FPN. Area ratio between the boxes of the corrupted model and the boxes of the fault-free model per layer injection when considering only SDC and Critical predictions. Weights FI

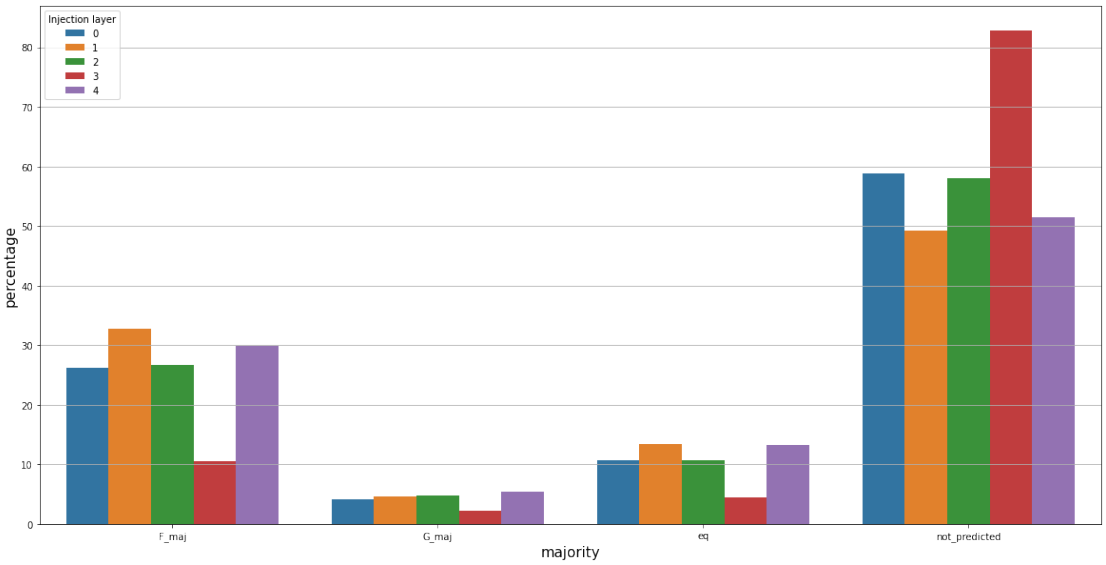


Figure A.7: SC Faster RCNN with FPN prediction majority - Weights FI. The percentage of cases in which the number of boxes predicted by the faulty SC Faster RCNN with FPN is smaller (F_maj), larger (G_maj) or equal (eq) than the number of boxes predicted by the corresponding golden model, or when the corrupted model made no predictions (not_predicted).

Bibliography

[1] Yoshitomo Matsubara, Ruihan Yang, Marco Levorato, and Stephan Mandt. «Supervised Compression for Resource-Constrained Edge Computing Systems». In: *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE, Jan. 2022. DOI: 10.1109/wacv51458.2022.00100. URL: https://doi.org/10.1109%5C%2Fwacv51458.2022.00100 (cit. on pp. 2, 16, 28, 34, 37, 41, 57, 67).

[2] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016 (cit. on p. 5).

[3] Li Fei-Fei and Ranjay Krishna. «Searching for Computer Vision North Stars». In: *Daedalus* 151.2 (May 2022), pp. 85–99. ISSN: 0011-5266. DOI: 10.1162/daed_a_01902. eprint: https://direct.mit.edu/daed/article-pdf/151/2/85/2060580/daed_a_01902.pdf. URL: https://doi.org/10.1162/daed%5C_a%5C_01902 (cit. on p. 8).

[4] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. «A Survey of Deep Learning-Based Object Detection». In: *IEEE Access* 7 (2019), pp. 128837–128868. DOI: 10.1109/ACCESS.2019.2939201 (cit. on p. 8).

[5] Jonathan Long, Evan Shelhamer, and Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2015. arXiv: 1411.4038 [cs.CV] (cit. on p. 8).

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. «U-net: Convolutional networks for biomedical image segmentation». In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer. 2015, pp. 234–241 (cit. on p. 8).

[7] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele.

«The cityscapes dataset for semantic urban scene understanding». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3213–3223 (cit. on p. 8).

[8] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV] (cit. on pp. 8, 9, 11, 43).

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV] (cit. on pp. 9–11).

[10] Olivia Weng et al. «Tailor: Altering Skip Connections for Resource-Efficient Inference». In: *ACM Transactions on Reconfigurable Technology and Systems* (2023) (cit. on p. 10).

[11] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. «Mobilenets: Efficient convolutional neural networks for mobile vision applications». In: *arXiv preprint arXiv:1704.04861* (2017) (cit. on pp. 10, 14).

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. arXiv: 1502.01852 [cs.CV] (cit. on p. 11).

[13] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV] (cit. on p. 11).

[14] Pablo Jimenez Mateo, Claudio Fiandrino, and Joerg Widmer. «Analysis of TCP Performance in 5G mm-Wave Mobile Networks». In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. 2019, pp. 1–7. DOI: 10.1109/ICC.2019.8761718 (cit. on p. 12).

[15] Menglei Zhang, Michele Polese, Marco Mezzavilla, Jing Zhu, Sundeep Rangan, Shivendra Panwar, and Michele Zorzi. «Will TCP Work in mmWave 5G Cellular Networks?» In: *IEEE Communications Magazine* 57.1 (2019), pp. 65–71. DOI: 10.1109/MCOM.2018.1701370 (cit. on p. 12).

[16] Yoshitomo Matsubara, Marco Levorato, and Francesco Restuccia. «Split computing and early exiting for deep learning applications: Survey and research challenges». In: *ACM Computing Surveys* 55.5 (2022), pp. 1–30 (cit. on p. 12).

[17] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. *BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks*. 2017. arXiv: 1709.01686 [cs.NE] (cit. on p. 12).

[18] Arian Bakhtiarnia, Nemanja Milošević, Qi Zhang, Dragana Bajović, and Alexandros Iosifidis. «Dynamic Split Computing for Efficient Deep EDGE Intelligence». In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2023, pp. 1–5. DOI: 10.1109/ICASSP49357.2023.10096914 (cit. on p. 13).

[19] Guangli Li, Lei Liu, Xueying Wang, Xiao Dong, Peng Zhao, and Xiaobing Feng. *Auto-tuning Neural Network Quantization Framework for Collaborative Inference Between the Cloud and Edge*. 2018. arXiv: 1812.06426 [cs.DC] (cit. on p. 14).

[20] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML] (cit. on p. 16).

[21] Muhammad Shafique, Mahum Naseer, Theocharis Theocharides, Christos Kyrkou, Onur Mutlu, Lois Orosa, and Jungwook Choi. «Robust Machine Learning Systems: Challenges,Current Trends, Perspectives, and the Road Ahead». In: *IEEE Design & Test* 37.2 (2020), pp. 30–57. DOI: 10.1109/MDAT.2020.2971217 (cit. on pp. 18, 26).

[22] Alberto Bosio et al. «Emerging Computing Devices: Challenges and Opportunities for Test and Reliability». In: *2021 IEEE European Test Symposium (ETS)*. 2021, pp. 1–10. DOI: 10.1109/ETS50041.2021.9465409 (cit. on p. 18).

[23] Josie E. Rodriguez Condia, Juan-David Guerrero-Balaguera, Fernando F. Dos Santos, Matteo Sonza Reorda, and Paolo Rech. «A Multi-level Approach to Evaluate the Impact of GPU Permanent Faults on CNN's Reliability». In: *2022 IEEE International Test Conference (ITC)*. 2022, pp. 278–287. DOI: 10.1109/ITC50671.2022.00036 (cit. on p. 18).

[24] Alfredo Benso and Paolo Prinetto. *Fault injection techniques and tools for embedded systems reliability evaluation*. Vol. 23. Springer Science & Business Media, 2003 (cit. on p. 19).

[25] Annachiara Ruospo, Ernesto Sanchez, Lucas Matana Luza, Luigi Dilillo, Marcello Traiola, and Alberto Bosio. «A Survey on Deep Learning Resilience Assessment Methodologies». In: *Computer* 56.2 (2023), pp. 57–66. DOI: 10.1109/MC.2022.3217841 (cit. on pp. 20, 23).

[26] Behzad Salami, Osman S. Unsal, and Adrian Cristal Kestelman. «On the Resilience of RTL NN Accelerators: Fault Characterization and Mitigation». In: *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. 2018, pp. 322–329. DOI: 10.1109/CAHPC.2018.8645906 (cit. on p. 23).

[27] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. «Ares: A framework for quantifying the resilience of deep neural networks». In: *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 2018, pp. 1–6. DOI: 10.1109/DAC.2018.8465834 (cit. on pp. 23, 24).

[28] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert. «Statistical fault injection: Quantified error and confidence». In: *2009 Design, Automation & Test in Europe Conference & Exhibition*. 2009, pp. 502–506. DOI: 10.1109/DATE.2009.5090716 (cit. on pp. 24, 29).

[29] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari. «PyTorchFI: A Runtime Perturbation Tool for DNNs». In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 2020, pp. 25–31 (cit. on p. 24).

[30] Zitao Chen, Niranjhana Narayanan, Bo Fang, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. *TensorFI: A Flexible Fault Injection Framework for TensorFlow Applications*. 2020. arXiv: 2004.01743 [cs.DC] (cit. on p. 24).

[31] Annachiara Ruospo, Angelo Balaara, Alberto Bosio, and Ernesto Sanchez. «A Pipelined Multi-Level Fault Injector for Deep Neural Networks». In: *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. 2020, pp. 1–6. DOI: 10.1109/DFT50435.2020.9250866 (cit. on p. 25).

[32] Yang Zheng, Zhenye Feng, Zheng Hu, and Ke Pei. «MindFI: A Fault Injection Tool for Reliability Assessment of MindSpore Applicacions». In: *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. 2021, pp. 235–238. DOI: 10.1109/ISSREW53611.2021.00068 (cit. on p. 25).

[33] G. Gavarini, A. Ruospo, and E. Sanchez. «SCI-FI: a Smart, aCcurate and unIntrusive Fault-Injector for Deep Neural Networks». In: *2023 IEEE European Test Symposium (ETS)*. 2023, pp. 1–6. DOI: 10.1109/ETS56758.2023.10173957 (cit. on p. 25).

[34] Abdulrahman Mahmoud et al. «Optimizing Selective Protection for CNN Resilience». In: *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. 2021, pp. 127–138. DOI: 10.1109/ISSRE52982.2021.00025 (cit. on p. 25).

[35] Alessio Colucci, Andreas Steininger, and Muhammad Shafique. *ISimDL: Importance Sampling-Driven Acceleration of Fault Injection Simulations for Evaluating the Robustness of Deep Learning*. 2023. arXiv: 2303.08035 [cs.LG] (cit. on p. 26).

[36] Niccolò Cavagnero, Fernando Dos Santos, Marco Ciccone, Giuseppe Averta, Tatiana Tommasi, and Paolo Rech. «Transient-Fault-Aware Design and Training to Enhance DNNs Reliability with Zero-Overhead». In: *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2022, pp. 1–7. DOI: 10.1109/IOLTS56730.2022.9897813 (cit. on pp. 27, 68).

[37] Luzhou Peng, Bowen Qiang, and Jiacheng Wu. «A Survey: Image Classification Models Based on Convolutional Neural Networks». In: *2022 14th International Conference on Computer Research and Development (ICCRD)*. 2022, pp. 291–298. DOI: 10.1109/ICCRD54409.2022.9730565 (cit. on p. 32).

[38] Andrew Howard et al. *Searching for MobileNetV3*. 2019. arXiv: 1905.02244 [cs.CV] (cit. on p. 35).

[39] Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. *Object Detection in 20 Years: A Survey*. 2023. arXiv: 1905.05055 [cs.CV] (cit. on p. 39).

[40] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV] (cit. on p. 41).

[41] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. «SSD: Single Shot MultiBox Detector». In: *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2. URL: https://doi.org/10.1007%5C%2F978-3-319-46448-0_2 (cit. on p. 42).

[42] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. «ImageNet: A large-scale hierarchical image database». In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848 (cit. on p. 57).