POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Fault Detection Metrics in Image Segmentation Neural Networks

Supervisors

Dr. Annachiara RUOSPO

Prof. Ernesto SANCHEZ

Candidate

Lorenzo FEZZA

April 2025

Abstract

Faults that occur on weights of semantic segmentation networks can significantly compromise the final result of the output mask. In particular determining the degree of such criticality is not a trivial task and, in some cases, it is also extremely crucial. For this purpose, various techniques have been developed to evaluate the output mask. The state of the art methods consist in the comparison between the faulty and faultless output of the network, based on two main segmentation metrics: mean intersection over union (mIoU) and pixel accuracy (PA). These methods, which turns out to be very effective, actually hides several problems and in particular, the main one is that, in real cases, there is no guarantee to have access to the faultless output. In addition, subjective thresholds used for the classification do not always provide the desired results. Other techniques rely on the usage of an auxiliary neural network, also subject to faults and their latency is not always sufficiently low to guarantee the real time requirement, which is a fundamental constraint in many fields such as the automotive one.

This thesis describes the application of state of the art methods on a specific image segmentation network and proposes a new metric, applicable in environments where real time constraint is mandatory, aimed at evaluating and identifying criticalities in segmentation masks, when the network is subject to hardware faults. The advantage introduced by this metric is that it is able to evaluate a single output without having access to the clean one and the temporal coherence across input frames is not required. The latter is a significant detail because masks are evaluated individually, without considering the previous frames, potentially affected by critical faults. The metric involves a statistical approach with an initial overhead that depends on the size of the dataset partition used to produce statistics on the clean network inferences. A labeled dataset is also proposed to evaluate results obtained with the developed metric, compared with the state of the art metrics used for this kind of classification.

The network employed is known as Fast-SCNN, an open source semantic segmentation network, developed in python with the PyTorch library, pre-trained and tested on Cityscapes. A fault injector is also used to simulate the faulty network by injecting stack-at faults on a specific bit of the weight. The output masks are then analyzed with the developed metric, composed by different units, each involved in identifying whether a fault is critical or not (binary task). Additionally, a deeper study is performed on each metric unit to investigate the impact on the final results and performances are compared to the state of the art methods, highlighting the fact that identifying a critical fault is a problem that is far from solved and that metrics actually available are not always reliable.

Acknowledgements

I wanted to thank all those people who have been by my side on this unbelievable and magical journey, people who have constantly supported me and believed in me even in the most difficult moments.

A special thanks goes to my family, who made all this possible, taking care of me and doing everything to give me access to the knowledge that I have acquired from when I was a child until today.

To my friends, who have made my entire life an unforgettable adventure and who brought color into my world, even when everything around me faded to gray.

From an academic point of view, I express my gratitude to all my professors, whose availability and patience have allowed me to develop strong skills across the board.

I would especially like to thank my supervisors, Annachiara Ruospo and Ernesto Sanchez, whose support has enriched me in countless ways, starting with the academic aspect, where they have been invaluable guides, helping me to overcome even the most difficult obstacles of this research, also allowing me to give vent to my creativity, to always explore new approaches and innovative methods, up to the human one, where they have always been available to create a beneficial environment, favorable to growth and learning.

Finally, I would like to thank Vittorio Turco, a very kind and enterprising PhD student who has been a constant source of support throughout every stage of this research.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
Artificial Intelligence

DNNs
Deep Neural Networks

CNN
Convolutional Neural Network

SDC
Silent Data Corruptions

ABFT
Algorithmic Based Fault Tolerance

DNA
Distribution of Neuron Activations

PA
Pixel Accuracy

mIoU
mean Intersection over Union

fps
frames per second

FI
Fault Injection

FOD

Faulty Output Dataset

PWM

Position Weight Matrix

SWM

Symmetry Weight Matrix

CO

Clean Output

CM

Confusion Matrix

FP

False Positive

FN

False Negative

DCF

Detection Cost Function

Chapter 1

Introduction

Technological development of the last few years allowed to design increasingly innovative Artificial Intelligence (AI) systems, which are able to learn to solve very complex tasks indeed, that go beyond the simple binary or multiple classification. Among these, image segmentation, which consists in the pixel wise annotation of an image, has become a task of common interest for different fields, like medical, security and automotive. However, it is precisely in these areas that reliability of the segmentation network becomes a primary requirement.

The analysis of the behavior of the network in presence of faults highlights how much these systems are not particularly reliable and resilient. For this reason it is necessary to improve the fault management and guarantee greater operational continuity. In the context of fault detection, for instance, the state-of-the-art methods include the usage of Pixel Accuracy (PA) [1], a metric which, as for the mean Intersection over Union (mIoU) [2], is widely adopted to evaluate the performance of semantic segmentation networks. However, these metrics require the use of the mask produced by the faultless network, which is not always guaranteed to be available. It would be possible to use additional segmentation networks that are assumed to be fault-free, but what ensures that they are truly faultless? Another option would be to train networks to classify the mask as either faulty or clean, but this would also lead to a circular issue, as it has been observed that even these networks are subject to critical faults [3] and additionally, the final output is not even interpretable. As a result, finding a metric that allows to evaluate semantic segmentation masks without the usage of a ground truth is still an open question.

Furthermore, it must be considered that semantic segmentation networks often require a real-time response as a fundamental constraint, therefore the metrics should also be efficient. A concrete example is given by the autonomous driving scenario, in which real-time urban scene recognition is a fundamental task and if a fault is critical for the network, elements such as roads, cars or people may not be

properly identified and this can trigger even catastrophic accidents.

However, to exhaustively access the reliability of such networks, it is necessary to inject one fault for each single bit for every weight and since parameters of current Deep Neural Networks (DNNs) have significantly increased, doing so is extremely expensive. To reduce analysis times, it is possible to perform statistical fault injection [2], which involves injecting a limited number of faults instead of all of them, ensuring an error margin in the accuracy of the analysis.

In this thesis, the behavior of the Fast-SCNN [4], a segmentation network trained on Cityscapes [5], is investigated through statistical fault injection. The approach involves the use of an additional dataset developed within this thesis, containing a series of masks obtained from the faulty network, with faults injected on bit 30, each manually labeled as critical or not. This dataset allows to have a general perspective on the behavior of the network and allows to evaluate the metric developed in the following sections. The metric is created with the aim of evaluating the quality of the mask, as for mIoU and PA and unlike these, it does not require the use of the clean mask during its use and relies on a statistical approach to establish thresholds defined in advance to identify potential anomalies.

As a whole, the thesis is organized as follows: the first section is aimed to describe the fundamental elements on which the research is based upon, including the task and related problems, starting from faults and going through the methods adopted to access the reliability of Neural Networks, concluding with the state of the art metrics employed to detect such faults. Subsequently, the proposed approach lays down the description of the new labeled dataset and the set of metrics designed to identify anomalies from the network output. Afterwards, the experimental results section contains a study of the developed metric with accuracy and risk that the usage of the latter reaches on the developed dataset. Results are then compared to the state of the art fault detection methods, concluding with a summary of the executed work and achieved results with a look towards future developments.

Chapter 2

Background

The following section provides an exhaustive explanation of the foundational elements that lay the groundwork for this thesis. The first subsection is aimed to describe the basic components of the analyzed system, starting from the task and going through the dataset and network adopted. The next subsection introduces hardware faults related to the network weights and their impact on the corresponding output. In addition, faults can critically compromise the final prediction and it is possible to identify and measure the degree of such damage with different methods. For this purpose, the following subsection describes metrics that are currently available, relying on mIoU and PA.

2.1 Image Segmentation

In the context of computer vision, a task refers to a specific problem which requires image processing and interpretation. More in general, it is a problem that a model is designed to solve. Neural networks are widely employed to address such tasks, learning complex patterns and representations from large datasets. These models, particularly deep learning architectures, have proven to be surprisingly powerful in solving such tasks.

The following subsections will provide a brief discussion of these basic components, such as task, dataset, and network employed in the research.

2.1.1 Task: Image Segmentation

Image segmentation is a computer vision task which consists in dividing a digital picture into distinct pixel clusters (or segments), in order to organize a complex image content into well-defined sections [6].

Since the purpose is to precisely detect distinct regions, this is a common task

for various fields [6] such as medical, where it is used for tumor detection or brain segmentation, automotive, allowing autonomous driving, as well as agriculture, surveillance systems and many others.

Image segmentation includes various categories [6], varying according to a specific objective, namely:

- **Semantic segmentation:** The simplest case, it consists in the pixel wise annotation of the image. A label, chosen from a list of different classes, is assigned to each image pixel. Classes can represent object categories which are separable in different instances (things), e.g. cars, animals or people, but also undivided entities (stuffs), e.g. sky, terrain or sea. The further subdivision into single instances, in this case, is not provided.
- **Instance segmentation:** It aims to detect only things, dividing them in single instances.
- **Panoptic segmentation:** It combines the semantic and instance segmentation tasks, in order to identify stuffs and things, further divided into single instances, resulting to be the most complex and detailed among them.

The focus of this thesis is **semantic segmentation**, which represents the base for many practical applications and one of the most consolidated and studied variants.

2.1.2 Dataset: Cityscapes

Cityscapes constitutes a widely used resource for semantic segmentation and urban scene comprehension, finding large application in image processing research and deep learning. Introduced in 2016, Cityscapes offers an extended collection of high resolution scenes taken from 50 European cities, including a variety of architectural, climatic and light conditions [5].

The dataset consists of 5000 high-quality images with pixel-level manual annotations and it divided into 3 different partitions, respectively:

- Training set: 2975 annotated images.
- Validation set: 500 annotated images.
- Test set: 1525 images without annotations.

In addition, other kinds of images with different forms of annotations are also available, supporting complementary applications to semantic segmentation.

The complexity of the dataset is given by the number of classes considered. In particular, it includes 30 different classes, categorized in 8 principal groups,

respectively flat, construction, nature, vehicle, sky, object, human, and void. Only 19 of them (listed in Figure 2.1) are actually available in the dataset, while the remaining ones, rarer with respect to the others, are merged into the void class (identified as -1). Figure 2.2 shows a Cityscape image with the corresponding annotation on the right.

Class	Void	Road	Sidewalk	Building	Wall	Fence	Pole	Traffic Light	Traffic Sign	Vegetation
Id	-1	0	1	2	3	4	5	6	7	8
Color										

Class	Terrain	Sky	Person	Rider	Car	Truck	Bus	Train	Motorcycle	Bicycle
Id	9	10	11	12	13	14	15	16	17	18
Color										

Figure 2.1: List of Cityscapes Classes

Figure 2.2: Cityscapes example of picture (on the left) with the corresponding ground truth (on the right) taken from the validation partition

2.1.3 Network: Fast-SCNN

Fast-SCNN [4] is a Convolutional Neural Network (CNN) employed for real-time semantic segmentation tasks. This network allows to process high resolution images (1024×2048 pixels) efficiently from the computational point of view and it requires low memory consumption. In particular, using a GPU Nvidia Titan Xp (Pascal), this network allows to process 123.5 frames per second (fps) with an accuracy, measured in terms of mIoU, of 68% on Cityscapes [4].

As shown in Figure 2.3, the architecture is combination of two structures:

- The **encoder-decoder** architecture, originally and specifically developed for image segmentation tasks, consists of an encoder, which deeply extracts

features information from the input image and a decoder which decodes it, providing the final result;

- The **two-branch** structure, which allows to obtain a notable increase in performance compared to traditional architecture, by extracting low and high level features in two separate branches and combining them into a single feature map, the one that is finally decoded.

Entirely, the architecture is divided in four principal components:

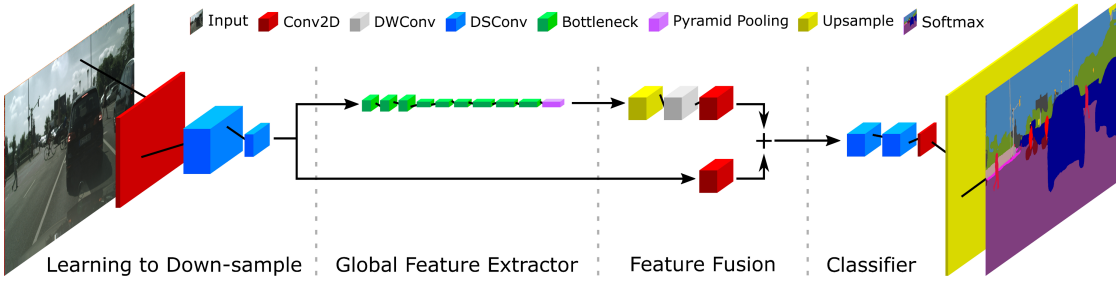


Figure 2.3: Fast-SCNN model representation from [4]

- **Learning to Downsample:** This first block deals with extracting, through three convolutional units, low level features, such as edges, gradient orientation, texture and color.
- **Global Feature Extractor:** Block where features are processed, it is made of the above mentioned two-branch: the first one is deeper, responsible to extract the global context of the image, while the other one, shallow, handles the extraction of the spatial information.
- **Feature Fusion Module:** It is used to merge global context and spatial information produced by the preceding block.
- **Classifier:** Adopted at the end of the network, it improves the accuracy of the output by exploiting three additional convolutional units, producing the final prediction.

The network is open source, available online pre-trained on Cityscapes with a mIoU of 54.84% and PA of 92.37%. Morover it is developed with the PyTorch framework and its weights are represented in the FP32 format. While this detail may seem supplementary at this stage, it becomes relevant in considerations discussed in the following section.

2.2 Hardware faults

The development of DNNs enabled their growing employment in critical use cases, proving to be extremely promising at solving a large variety of tasks. However, DNNs have been developed focusing on performance and efficiency, neglecting reliability [3], which is a fundamental constraint for such fields. In particular, for reliability is intended the probability of a system to properly behave in presence of faults. For this reason, the latter cannot be overlooked and they are currently the subject of studies aimed at enabling the proper use of the network in the real world, minimizing risks.

Faults are anomalies in a system that potentially lead to errors if the output is different from the expected one. In this case, faults are named *active* [7], while if the fault is present but it does not trigger any error, it is referred to as *dormant* [7]. When the error prevents the system from performing its intended task, it causes a failure [7].

Faults arise from several factors and manifest in different ways [3]: process variations, timing errors, but also voltage scaling and near-threshold computing are some examples. The taxonomy proposed by Mittal et. al. [3] reports a classification based on the source and the types of faults. Depending on the number of bits affected, a fault can be defined as single-bit or multi-bit. Another taxonomy [7] is based on the time interval in which the fault affects the network. A fault is considered **permanent** if it persists in the network for an extended and indefinite period. In contrast, a fault is called **transient**, if it lasts for a short time. One of the most studied ones is the stack-at, a permanent fault characterized by data or control lines fixed at a specific value (e.g., 0 or 1). This kind of fault is able to emulate, with sufficient accuracy, transistor and interconnection level irregularities [8], allowing to study the behavior of the network in presence of faults, without considering the physical structure of the network. In particular, considering the abstract model of a neuron, a stack-at-0 or a stack-at-1 model interconnections and synaptic weights faults, while a stack-at-value reproduces faults in the summation and the activation function.

This subsection contains initially a description of the DNNs model parameters, where stack-at faults studied in this research take place, with a special emphasis on the weight representation and specific characteristics that make the network more or less robust. To study faults and the behavior of the network in their presence, it is possible to simulate them in different ways, as discussed subsequently. Finally, it is explained how the number of DNNs parameters impacts the study of its reliability and how statistical fault injection [9] comes to the rescue.

2.2.1 Weight Encoding and Precision Details

DNNs consist of millions of interconnected weights organized in various ways to process the input and generate the final output. Weights are contemporary the key and most basic components of DNNs: they constitute parameters that models are able to learn during the training process to finally perform inferences during the test phase and it is precisely on these parameters that permanent faults possibly occur, compromising the reliability of the system.

Depending on how the network is designed, it is possible to adopt different representations for them, as shown in Figure 2.4.

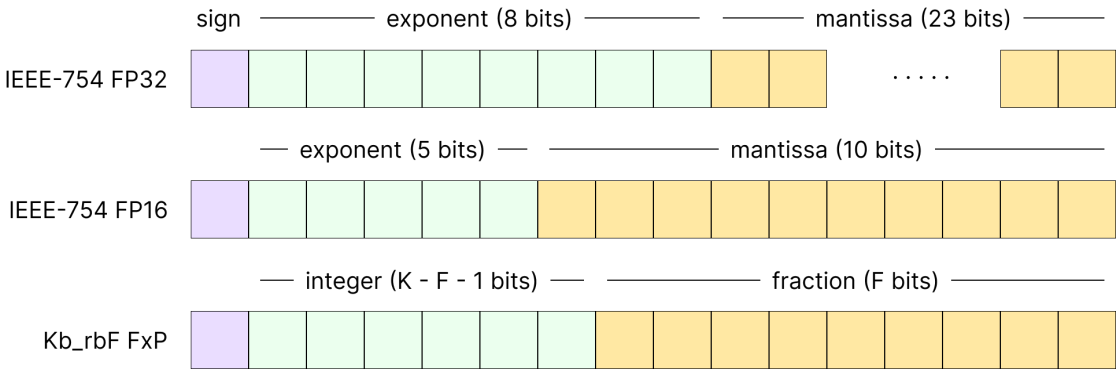


Figure 2.4: Different weight representations

For each representation, the parameter can take on different value ranges. As noted in [3], the choice of format directly influences the network reliability, just as other components—such as activation functions, convolutional layers, and datasets—do. However, these aspects are not the focus of this thesis. Nonetheless, they provide useful context for understanding why faults affecting certain bits may have a more critical impact than others.

2.2.2 Fault Injection

To study the reliability of a network, it is possible to analyze the behavior the model in presence of faults by simulating them. This process, known as Fault Injection (FI), can be performed using various approaches [10], divided into three macro categories:

- **Simulation based FI**: It consists in simulating faults without considering the physical device on which the network is employed. This approach is further distinguished in two abstraction levels:

  - **Software level**: This process involves injecting the faults on the high-level model of DNNs via software, independently from the hardware where the model is adopted, not considered in the assessment. Using frameworks such as PyTorch it is possible to develop an injector in a simple and robust way, allowing to replicate the experiments deterministically. However, results are not particularly accurate since the hardware is completely not taken into account;
  - **Hardware level**: Faults are injected by simulating the target hardware on which the network is employed at Register Transfer Level or at Gate Level. Differently from Software level FI, this process not only involves the Application level, but also the Architecture of the network at Hardware level;

- **Platform based FI**: The process is similar to the Simulation based FI, but injections are directly performed on the platform (GPUs, CPUs or FPGAs) on which the network is employed. As for the Hardware level FI, this process takes into account the Hardware abstraction level, but faults are injected physically on the Hardware and not simulated in the architecture.

- **Radiation based FI**: It requires specific facilities and devices to simulate the external conditions of the application network environment. It consists in radiating neuron particles on the physical hardware on which the network is adopted without considering the application level. Results of this assessment are particularly accurate but experiments are more difficult to reproduce and sepcific devices are required for these experimetns. In addition, model components which are not exposed to radiations cannot be injected.

Figure 2.5 reveals a visual representation of the aforementioned methods, while Table 2.1, likewise taken from [10], shows in summary advantages and disadvantages introduced by each method is terms of different metrics.

Although it is more computationally expensive and requires additional techniques to ensure acceptable results (e.g. statistical fault injection), software level fault injection is much more practical, since the equipment required for beam-radiation is available in few laboratories and if there are model components that are not exposed to radiations, there is no guarantee to obtain significant results.

The impact of such faults vary significantly depending on multiple factors, including the number of bits affected, the specific weights involved, the layers in which the faults occur, and the overall architecture of the network. In this study, stuck-at faults are injected by flipping a single bit at a time (single-bit fault) in the weights of a specific layer. Each fault involves forcing a bit to the inverse of its original value, with only one bit altered per injection, to isolate its effect on the network behavior.

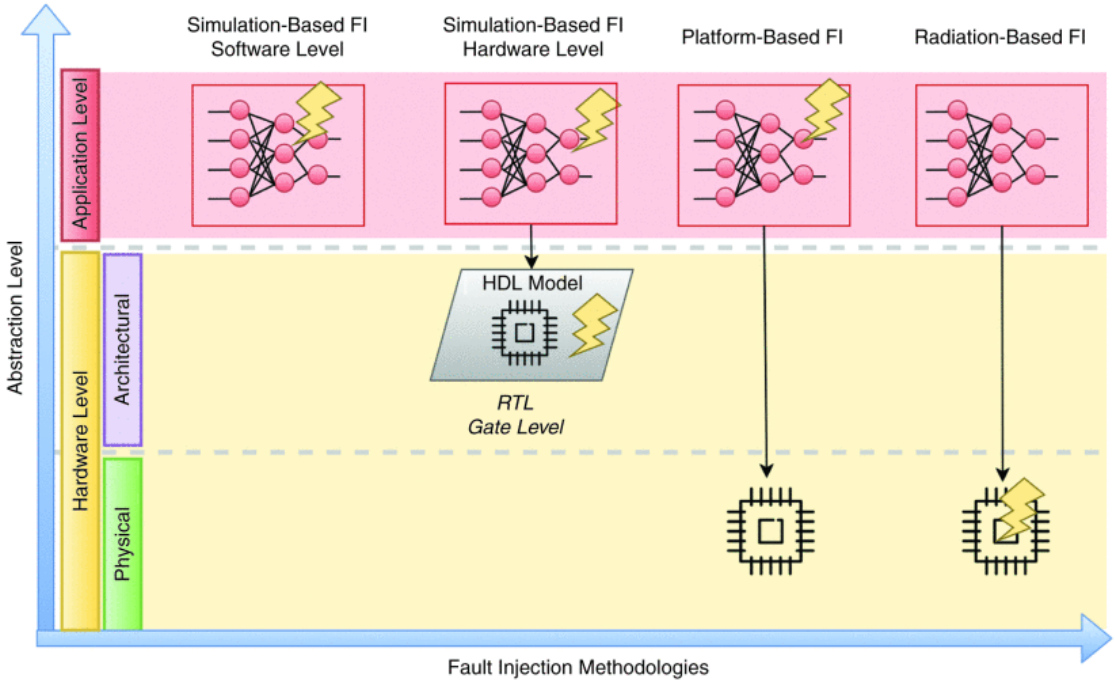


Figure 2.5: A taxonomy for the FI methodologies developed for the resilience assessment of DNNs and DNN-based systems from [10]

Table 2.1: A comparison among the different FI methodologies from [10]

Metric	**Simulation-Based Software Level**	**Simulation-Based Hardware Level**	**Platform Based**	**Radiation Based**
Cost	Low	Low	Medium–high	High
Development effort	Medium–high	Medium–high	High	Low–medium
Exactness*	Low	Medium–high	Low–medium	Very high
Controllability	High	High	Medium	Low
Observability	High	Medium**	Low**	Low**
Repeatability	High	High	High	Medium–low
Early availability	High	Medium	High	Medium–low
FI time	Low	High	Medium–low	Low
Principal advantages	Cheap and fast	Good FI exactness	Portability	Best FI exactness; realistic
Principal drawbacks	Low FI exactness	Time consuming; the HDL must be available	Limited FI exactness	Expensive

*Closeness to reality.

**The observability depends on the complexity of the hardware, which is used for the implementation of the FI process.

2.2.3 Statistical Fault Injection

Technological advancements have enabled the development of increasingly deeper CNNs, characterized by a significantly higher number of model parameters. The exhaustive study of the reliability of a model would necessitate the injection of a particularly high number of faults, depending from the representation of the weights, and how many of them constitute the network. Specifically, the total number of faults to consider for an exhaustive FI campaign is given by the trivial formulation $N = N_p \cdot b \cdot 2$, where N_p is the total number of model parameters and b is the number of bits derived from the weight representation. For example, a network like ResNet-20, consisting of 268346 parameters, would demand to take into account 17174144 faults [9]. Unfortunately, executing a large number of software FIs takes a significant amount of time, memory and, in some cases, it may become impractical.

However it is possible to obtain results close to an exhaustive analysis by using different techniques, thus significantly shortening the times. One such method, Statistical FI [9], involves, under some particular assumptions, the injection of a specific sample n of the entire pool N ($n << N$), with the guarantee of having a maximum error margin e.

As mentioned in [9], the sample size n is determined according to statistical inference, in order to generalize results of the sample n to the entire population N. In summary, a maximum error margin e is chosen on a specific metric, assuming that, for the total population N, the actual estimate has mean μ and variance σ^2 and, for an estimation x, the mean is μ_x and variance σ_x^2. Since the entire population N is finite, it is possible to apply a correction factor, that keeps into account the fact that the computed estimate is given by only a sample of the whole population and finally n is chosen according to the assumptions:

$$e = t * \frac{\sigma}{\sqrt{n}} \rightarrow n = \frac{N}{1 + e^2 \cdot \frac{N-1}{t^2 \cdot p*(1-p)}} \tag{2.1}$$

where t is a constant depending on desired confidence level and p is probability of a successful event ($0 \leq p \leq 1$). Selecting a value of 0.5 for the a-priori estimate p yields the maximum sample size n. However, in specific contexts, it is possible to choose a different value for p, leading to a reduction in the required sample size n.

Formulation 2.1 ([9], [2]) is based upon different assumptions:

- Each trial, corresponding to a single fault injection, follows a Bernoulli distribution $X \sim B(p)$, with p, the probability of the fault to become a failure;
- For n repeated experiments, the discrete random variable X follows a Binomial distribution $X \sim B(n, p)$;
- The Central Limit Theorem, which states that the distribution of the sum (or average) of a large number n of independent, identically distributed random

variables tends to a normal distribution, enables the computation of the variance $\sigma^2 = n \cdot p \cdot (1 - p)$, the correction factor used in formulation 2.1 to compute n.

When assuming the binomial distribution, it is required that each trial satisfies certain conditions [9]:

1. Each trial (fault injection) must result in one of two possible outcomes (e.g., Critical/Non-Critical)
2. The outcomes of the trials are mutually independent
3. The number n of trials is fixed
4. The probability of an outcome remains constant for each trial

Unfortunately, in CNNs, the fourth assumption does not hold, since, as seen in section 2.2.1 and deeply in [3], the probability that a fault becomes a critical failure, p, is influenced by many different factors. However, despite this method does not allow to identify the weakest units of the network, which besides is not the goal of this thesis, it is able to outline the general behavior of the faulty network, by considering it as black box [2].

2.3 Fault Detection

As seen in the previous sections, fault tolerance and the study of reliability of Neural Networks is a fundamental task, which aims to enable their employment in that specific cases where faults are particularly expensive. In the last few years, a big variety of methods, oriented to fault detection and mitigation, have been developed to access the reliability of such networks.

Recently, Turco et. al. [11] exploit metrics to evaluate the output feature maps from different Neural Network layers, in order to predict the potential impact of anomalies at inference time. By analyzing feature maps, it is possible to detect faults early, preventing their propagation, thus mitigating errors. This idea of prediction and prevention is also present in another approach [12], that focuses on Silent Data Corruptions (SDC), faults which remain silent until they occur in an error. The method consists in profiling the Distribution of Neuron Activations (DNA) from the faultless network, allowing to detect and mitigate the error from traces left by the SDC.

Robustness and resiliency of DNNs are highly investigated in [13], where clipping the activation function output into well defined ranges reveals to improve the error resilience efficiently.

Moreover the output of a CNN is provided by a series of matrix operations, which can be parallelized on specific hardware, allowing to obtain significant improvements in terms of computational cost. However, as explained in [14], improvements introduced by these high performance systems make matrices operations less robust. Thanks to the usage of Algorithmic Based Fault Tolerance (ABFT) [14], a method developed in 1984 and still in use today, it is possible to make these operations more robust without negatively affect the hardware performance by detecting and masking permanent and transient hardware failures efficiently. In addition, checksum-based ABFT is deeply investigated in [15], where the method is compared with recomputation for error correction, revealing that multiple error correction negatively impacts performances, evaluated in terms of computation capacity.

However, despite these advancements, fault detection for image segmentation tasks is still an area of early research. Actually the state of the art involves metrics such as mIoU and PA, widely adopted in evaluating the quality of image segmentation maps (masks). These metrics, combined with specific thresholds, allow to classify faults into different categories. The two metrics are described in the following subsections, while criteria and thresholds to detect faults are listed immediately after.

2.3.1 Pixel Accuracy

The metric PA allows to evaluate a predicted segmentation mask, by computing the percentage of pixels classified with the same label of a mask deemed as a ground truth. Simply put, it is the number of pixels correctly predicted over the total number of pixels.

As denoted in [2], achieving high PA is not always a reliable indicator of segmentation quality, particularly in cases of class imbalance, where one or a subset of classes dominates the mask over others. To better understand this problem, consider for example to have a model that segments brain scans to detect tumors. If the model predicts a mask in which no pixels are classified as tumors, but in reality the tumor exists and occupies only 1% of the total mask, then the pixel accuracy would be 99%, but the tumor mass would not be detected.

2.3.2 Mean Intersection over Union

The mIoU is another metric used to quantify the discrepancy between two semantic segmentation masks. The procedure involves calculating, for each class, the intersection of pixel regions having the same label in both the predicted mask (P_c) and the ground truth mask (G_c), dividing each value by the union of the respective regions:

$$IoU_c = \frac{P_c \cap G_c}{P_c \cup G_c} \tag{2.2}$$

The final value is then computed by averaging the IoU scores across all the classes. The mIoU of the last example (present in section 2.3.1) drops drastically to the 50 %, proving to be more sensitive to prediction errors and less vulnerable to class imbalance.

Both metrics are widely applied to measure the performance of segmentation networks, but they are also adopted in [1] and [2] to classify the impact of a fault on the final prediction.

Taxonomy of [1] is reported in Figure 2.7 and the method involves, only in the first step, the entire network output, to check whether the fault is masked or not. If the output is not bit-wise equivalent to the clean (faultless) one, the final mask is compared with the corresponding clean mask in terms of PA.

Differently, the classification method proposed in [2] relies on the computation of the difference between mIoUs of predicted and clean mask, as shown in Figure 2.6. This metric not only requires the faultless network mask, but also the ground truth with which the image should be properly labeled.

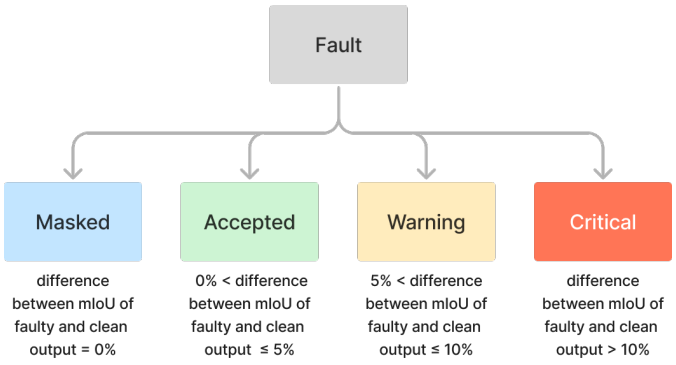


Figure 2.6: Taxonomy of semantic segmentation faults proposed in [2]

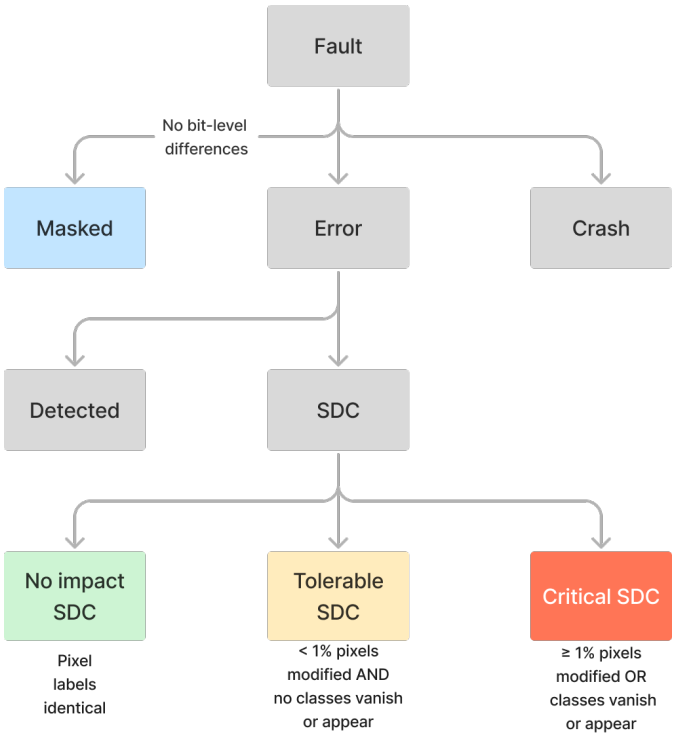


Figure 2.7: Taxonomy of semantic segmentation faults proposed in [1]

Chapter 3

Proposed Approach

The following chapter contains a detailed description of the method adopted to identify critical faults from the masks produced by segmentation networks.

The first section describes the Faulty Output Dataset (FOD), developed as part of this thesis. This dataset contains masks obtained from the faulty network, providing a general overview of the impact that bit 30 stack-at faults have on the network masks. Each mask is manually labeled with one of two categories, respectively critical or non-critical, allowing to evaluate and compare results obtained with different fault detection methods. In particular, the next subsection focuses on the developed technique, which aims to detect critical faults through the use of four units, each one employed for the analysis of specific characteristics of the mask in an interpretable and efficient way. Note that unlike the state-of-the-art metrics, this technique does not require the use of a ground truth to be computed, but in order to properly work, it relies on a setup phase, in which a statistical approach allows to set boundaries used to identify anomalies in a precise manner. This profiling process becomes necessary to understand the general behavior of the network in standard conditions.

3.1 Faulty Output Dataset

Depending on the input image and the fault injected inside the network, the final output mask can be labeled as critical or not, since some faults can be masked by mathematics operations through the convolutional layers, without having grave impacts on the final outcome [3]. To exhaustively evaluate the behavior of the metric, it should be known a priori whether all the faulty outputs are corrupted, but injection of all the possible faults on all the bits of each weight in every layer of the network for the whole dataset and the corresponding manual labelling (if critical or not) is unfeasible. Therefore, a reduced number of faults and partition

of the Cityscapes dataset are employed to design FOD. Since the purpose of this thesis is the metric evaluation and not the pure study of the vulnerabilities of the architecture, the network is injected with 8439 stack-at-params faults at bit 30, filtered from a fault list generated through statistical fault injection. The choice of this particular bit traces back to the fact that weights of this network are represented in FP32 and a change in that position drastically changes the parameter value, narrowing the search field, so as to obtain a critical output with more likelihood. As input for the network, only the validation set is selected, since it represents the smallest partition of Cityscapes, characterized by the highest variability of scenes. In addition, the ground truth is available for additional analysis, in contrast to the test set, which instead contains only unsegmented images. However, manually classify 8439×500 faulty outputs is expensive as well, for this reason a random sample of 68001 couples <image, fault> are selected and labeled. The sample still contains all the validation set frames and each fault of the statistical FI fault list. Some examples of masks present in FOD are shown in Table 3.1.

Note that PA and mIoU are not used for the labeling process, even if they are part of the dataset and comparing the labels with the state-of-the-art methods, they do not always match because the thresholds adopted in [1] and [2] are particularly stringent, classifying uncompromised outputs as critical and, similarly, failing to classify critical ones correctly. Results of this comparison are present in chapter 4.

In the following section, specifications of FOD are presented in more detail, supported by a brief statistical description.

3.1.1 Dataset Overview

The dataset is formed by the following columns:

- **Frame**: It is an integer value which identifies the input image index of the Cityscapes validation partition;
- **Injection**: It is an integer identifier assigned to a stack-at fault;
- **Layer**: It specifies the Fast-SCNN layer where the fault injection takes place;
- **TensorIndex**: It is a tuple of 4 values identifying the position of the weight inside the layer;
- **Bit**: It is an integer value identifying the bit to flip in the stack-at fault injection;
- **mIoU**: Mean interseciton over union computed between the clean and faulty output;
- **PA**: Pixel accuracy computed between the clean and faulty output;

Table 3.1: Some examples of FOD entries (only <frame, fault>, mIoU and PA for better readability of the table) with faulty mask compared to the clean one; C states for Critical, while NC states for Non-Critical

Frame	Injection	mIoU	PA	Label	Clean	Faulty
479	595	0.02	0.19	C		
295	9729	0.00	0.08	C		
331	7844	0.02	0.11	C		
10	1184	0.01	0.04	C		
191	12040	0.99	1.00	NC		
394	6867	0.02	0.18	C		
422	10192	0.01	0.02	C		
203	13188	0.01	0.05	C		
236	191	1.00	1.00	NC		
361	12499	0.02	0.24	C		

- **Label**: Integer/binary value representing critical or not critical (binary task), manually attributed to the pair <fault, frame>.

The columns relating to fault and frame allow anyone to reproduce the fault in a deterministic way, while the rest of the data is supportive and usable for the final evaluation.

FOD dataset is unbalanced, since the percentage of non-critical entries is only 11.90%, against the 88.10% labeled as critical.

Regarding the class distributions in the faulty masks, a portion of 43.07% of the dataset consists of masks where all pixels share the same value and for the 99.53% of them, this value is referred to the class 'road'. The remaining 56.93% is made of critical and non-critical masks with higher variability in the class distributions. Excluding the 43.07% portion, Figure 3.1 shows the variability of FOD masks. In particular, each grouping is defined basing on the tuple: number of unique classes contained in the mask (from 2 to 9) and the top 3 classes with largest number of occupied pixels.

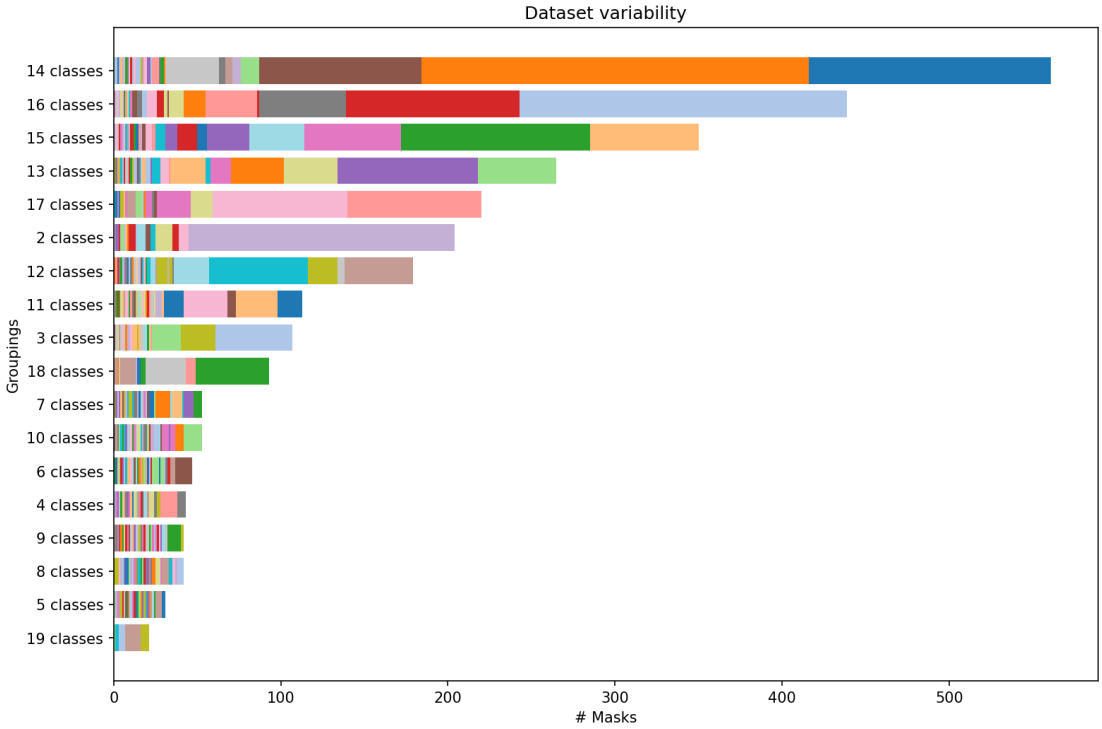


Figure 3.1: FOD masks variability representation: on x-axis the number of masks characterized by groupings defined basing on the tuple <number of unique classes contained in the mask (from 2 to 9), top 3 classes with largest number of occupied pixels>

3.2 Metrics

The overall metric combines four different methods, each one employed in the extraction of a different feature. The set of information obtained from each unit allows to evaluate the spatial coherence at the level of surfaces occupied by the individual classes, the positions they assume and their degree of symmetry, as well as the regularity of the contours between things and stuffs. Fundamental requirements are that the features must be extracted in real time and in a reliable and interpretable way. Although neural networks allow to reach very low latency times, the outputs are result of deep and complex interconnections between the weights of the network, so they are not easily interpretable and as explained in [3], they are far from reliable, given that a single change in a bit of a weight can change the final result. Therefore, the proposed extraction algorithms are particularly efficient and straightforward. Furthermore, they do not involve the use of input or golden mask, in contrast to the state of the art metrics, since there is no guarantee of having access to these data outside the context of reliability studies. The following section provides a detailed description of each method.

3.2.1 Area

The surface covered by a class in an image segmentation mask is bearishly computed with the sum of pixels labeled with that specific class. This feature does not represent a strong indicator of criticality, since the information acquired takes into account only the number of occupied pixels and although in some domains things and stuffs occupy well-defined spaces without too many variations, in other cases this assumption does not hold. However, assuming that the network is trained and used in an environment where the surfaces of the classes are distributed in a sufficiently similar way, this method is able to identify anomalies in a very simple and efficient manner.

3.2.2 Position

This feature is extracted by computing the element-wise product of the class mask with a Position Weight Matrix (PWM) and the final result is given by the summation of all the product values. The generic process is described in Figure 3.2.

PWM is a particular data structure where each pixel is assigned to one or more values which represent the position weight. There are infinitely many possible combinations of values that can be used to produce this data structure and the choice of construction method comes with different advantages and disadvantages. A simple technique consists in the usage of the same PWM, built with incremental

weights that vary from 1 to $H \times W$ (Figure 3.3), with H and W dimensions of the mask.

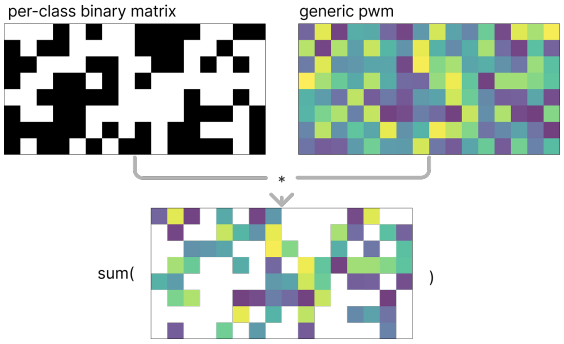


Figure 3.2: Position feature extraction process

When high resolution is required, the pixel count is significantly large, therefore, the maximum value achievable if a class is assigned to each pixel of the mask (worst case) is given by the Gauss formula $\frac{(H \times W) \cdot (H \times W + 1)}{2}$. For this reason, depending on the representation used for the final (integer) value, overflow cases must be taken into account.

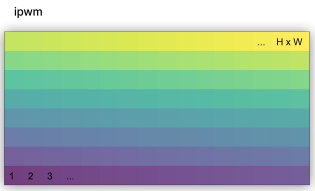


Figure 3.3: Incremental PWM

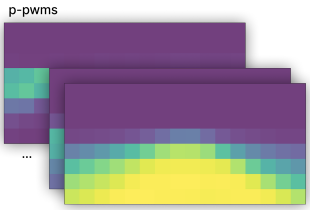


Figure 3.4: Probabilistic PWMs using probability to find the class in a specific pixel (p)

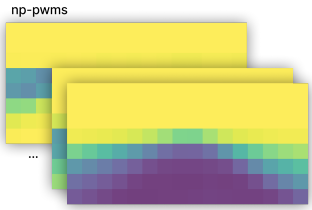


Figure 3.5: Probabilistic PWMs using probability to **not** find the class in a specific pixel ($1 - p$)

The shortcoming of the incremental PWM is given by the fact that summing many elements in low values cells is equivalent to adding few elements placed in high values cells, located in opposite positions inside the PWM. Even so, whenever a fault is critical for the network, if the position value is not anomalous for one class, it is not necessarily the same for the other ones. Nonetheless, the computational cost to produce this matrix is extremely low, no memory is required to store it persistently and its adoption requires products that does not involve floating point operations.

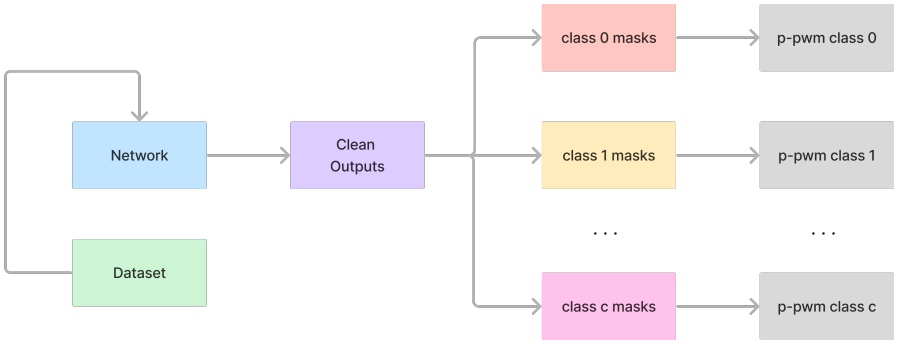


Figure 3.6: Probabilistic PWMs computation

Another method consists in the usage of multiple PWMs depending on how much memory is available. Using different PWMs allows to take into account the probability of having a class in a specific pixel-position. This method involves c different PWMs (where c is the number of assignable classes) and for each cell of the per-class PWM, the probability to find or not that class in that specific pixel is assigned. These probability values can be chosen ad-hoc, based on the prior knowledge about the position of a certain class, or they can be gathered with a statistical approach that involves computing the probabilities from the clean output over a dataset partition. Some examples are shown in Figures 3.4, 3.5, while the entire gathering process is shown in Figure 3.6.

The computation of the final value is similar to the incremental PWM, but in this case the product is computed for each class with the corresponding probabilistic PWM and since the values are floating point, products and summations are costlier and if the prior knowledge is not available, an additional overhead to obtain the probability values is needed before the position value computation.

3.2.3 Symmetries

As for the position unit, also symmetries are exploited through the usage of a single Symmetry Weight Matrix (SWM) or multiple SWMs. The feature extraction process requires the following steps for each class:

1. extract the class mask from the overall one obtaining a binary matrix
2. split the mask in half (horizontally or vertically, depending on the symmetries exploited)
3. flip the right half or the bottom half and overlap it on the corresponding half, doing an element-wise logic-and

4. compute the element-wise product of the half matrix with the corresponding SWM

5. sum all the elements in a single value

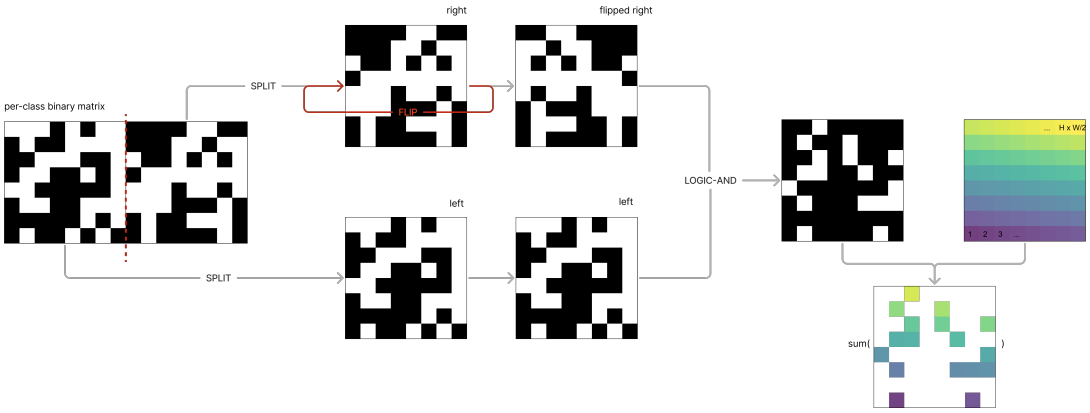


Figure 3.7: Vertical symmetry extraction process with incremental vertical SWM

The overall process to exploit vertical symmetry is shown in Figure 3.7 and the equivalent is possible for horizontal symmetry with a longitudinal split. Note that this process is similar to the above mentioned in section 3.2.2, but it requires three additional steps, involving the mask splitting, flipping and logic-and.

Regarding the SWMs, there are several alternatives. As for the position metric, an efficient and basic method consists in creating two simple matrices: one, for horizontal symmetry, of size $\frac{H}{2} \times W$ with incremental integer values from 1 to $\frac{H}{2} \times W$, the other one, to evaluate vertical symmetries, of shape $H \times \frac{W}{2}$, always with incremental values. To take into account the per-class-pixel symmetry probability, it is also possible to create probabilistic SWMs for each single class in which each cell is associated with the probability that a pixel of a class is horizontally (or vertically) symmetric. The process is same described in 3.6 for the positions, with the only difference that the probability is computed considering the vertically and horizontally folded masks computed immediately after the logic-and with the technique described in Figure 3.7. The higher the value is, the higher the class is symmetric, taking into account the pixel position.

3.2.4 Right Angles

Generally, segmentation networks produce masks, whose shapes have a certain degree of regularity. Measuring such feature is not a trivial task and there are various methods to do it. One possible way is to extract a pseudo-contour which

separates groups of pixels labeled with the same class to those others labeled with another class and count the number of right angles identified on this contour.

There are several ways to construct this pseudo-contour. The method adopted in this thesis involves a simple mask mapping from the predicted mask P to a new matrix M described by the following formula:

$$m_{i,j} = \begin{cases} 1 & \text{if } p_{i,j} \neq p_{i+1,j} \vee p_{i,j} \neq p_{i,j+1} \\ 0 & \text{otherwise} \end{cases}, \forall\, i, j \in H \times W \tag{3.1}$$

where $m_{i,j}$ is the mapped pixel in position i, j, with $i \in \{0, \ldots, H-1\}$, $\forall\, j \in \{0, \ldots, W-1\}$ and H, W dimensions of the network mask, while p is the mask pixel.

The final number of right angles, instead, is computed through the following steps:

1. from the contour mask M, extract two matrices: the first containing the segments of at least n pixels horizontally aligned, and the second one equivalent with at least n pixels vertically aligned

2. compute the intersection of masks obtained in steps 1

3. compute the sum of all the intersections (the final value is given by summing the mask from step 2)

The value obtained by this metric represents only an approximate but efficient count of the number of right angles formed by at least n aligned pixels. In addition, the final value depends on the selected n parameter. A possible way to set this value ($n_{min} = 3$) is comparing different performances on a specific partition of a dataset of interest.

3.3 Thresholds computation

The decision process is made possible with the usage of thresholds used to define whether a fault is critical or not. A metric value is considered anomalous if it is not contained within a lower and upper bound defined for that metric. These thresholds can be set based on prior knowledge, when available, but in more complex situations, when this data are not accessible, the simplest strategy relies on a statistical approach, described in Figure 3.8, in which a set of images large enough to capture greater variability in the input scenes is selected for processing.

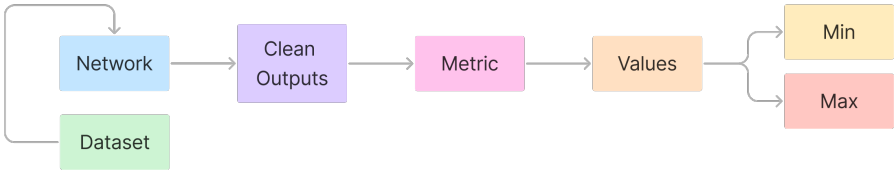


Figure 3.8: Thresholds computation (minimum and maximum boundaries) process using metric values retrieved from Clean Outputs from the Golden (Faultless) Network processing a set of data

Images are processed through the faultless network and minimum and maximum values are computed and set as boundaries in order to outline the standard behavior of the network.

Chapter 4

Experimental Results

This section is aimed to describe different possible set-ups of the developed metric and it contains the analysis of experimental results obtained applying each method to the clean masks and the FOD, in addition to a comparison with state of the art performance.

Stuck-at faults of this thesis are simulated through the usage of SFIadvanced-models [16], a software fault injector, originally adopted for image classification and appropriately modified for image segmentation purposes.

Regarding materials and resources adopted in this section: experiments have been executed remotely on a GPU NVIDIA GeForce RTX 3060 Ti; the software is written in Python and the framework used for the network, Fast-SCNN, is PyTorch, compatible with the fault injector adopted. The usage of Numpy, Matplotlib, Seaborn and Pandas libraries allowed to simplify loading, saving and graphic representation of results. The dataset on which experiments were performed is Cityscapes, on which the pre-trained network obtains a performance of 54.84% in terms of mIoU and 92.37% for PA over the Cityscapes validation partition.

Each component is first evaluated individually, considering multiple possible configurations to assess their specific contributions and costs. Unit by unit, performance variations are analyzed incrementally, showing how each unit affects the overall system. Finally, the metric is compared with techniques using mIoU and PA, in order to identify differences, vulnerabilities and advantages introduced by the proposed metric.

4.1 Metric Set Up

As described in chapter 3, the metric requires a set-up process to operate. In particular, this procedure involves the computation of statistics, to define discriminant thresholds used to identify anomalies in the segmentation mask and probabilities,

when probabilistic weight matrices for position and symmetry metrics are employed. Note that since the Cityscapes validation set is adopted in the FOD dataset, which is used to evaluate the performance of each metric, this partition is never used for any parameter selection and threshold computation.

The results for each individual metric, with various configurations, are explored section by section. Then, from each unit, the final metric is progressively built, with results shown incrementally at the end.

4.1.1 Area

Starting from the first unit, per class area statistics are computed over the Clean Output (CO) of the Cityscapes partitions in order to compute boundaries used to discriminate critical faults. A visual representation of per-class areas distribution is present in the bounding box plots in Figures 4.1 and 4.2. The same representation is also used for the other metrics in order to better understand each feature distribution. Table 4.1 shows the boundaries (as percentages of occupied pixels to facilitate understanding) employed. However, these are not the actual values used, since comparing directly the number of occupied pixels is sufficient. Boundaries vary for several reasons, e.g. training set size and scenes differ from the test set ones. In addition, the training partition is used to train the network, in contrast with the test set which contains unseen data.

Different performances and costs of using the two partitions are shown in Table 4.2 and the corresponding Confusion Matrix (CM) of each one is shown respectively in Figures 4.3 and 4.4. For the latter, a positive value corresponds to a Critical faulty output.

Comparing the performances obtained with the two partitions, using the training one turns out to be more suitable, since, although the accuracy with the test set is higher, the corresponding costs on COs are sufficiently large to make the test boundaries less effective. Furthermore, considering False Positive (FP) and False Negative (FN), the usage of the test set causes an increase of almost 10 times of FP values against a 1.7-fold decrease in FN.

4.1.2 Position

For the position unit, threshold computation is the same: feature extraction method is applied on COs over a dataset partition and maximum and minimum boundaries are computed accordingly. The method requires usage of one or multiple PWMs and each configuration comes with its own pros and cons.

Applying the incremental PWM over training and test partitions (as made for the area unit), allows to achieve different outcomes, as shown in Table 4.3 and Figures 4.5 and 4.6.

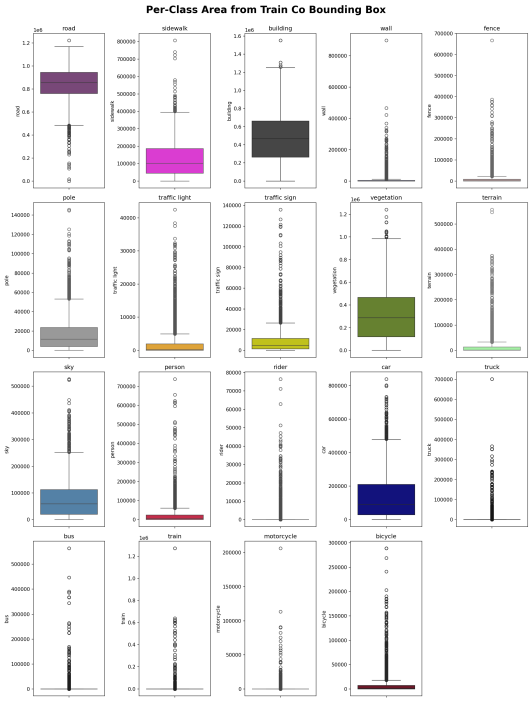


Figure 4.1: Per-class area bounding box plots with statistics from COs from the training partition of Cityscapes

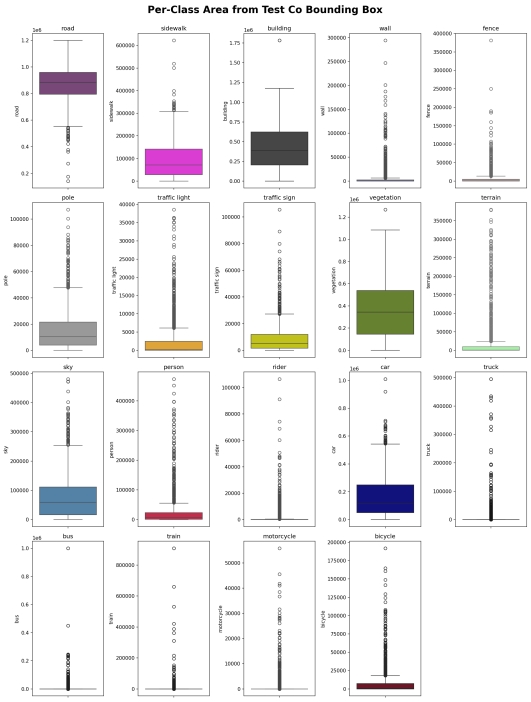


Figure 4.2: Per-class area bounding box plots with statistics from COs from the test partition of Cityscapes

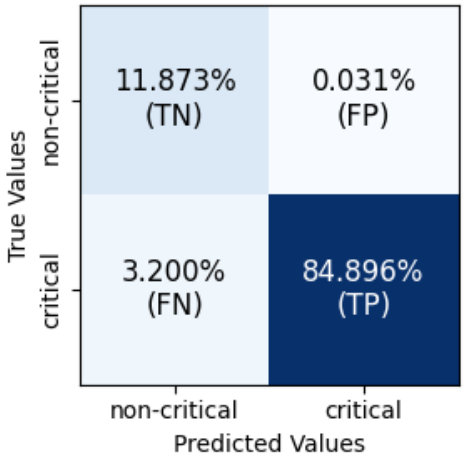


Figure 4.3: CM of the area unit with threshold made from training partition

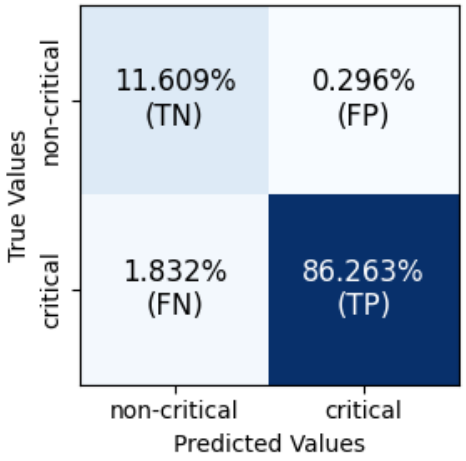


Figure 4.4: CM of the area unit with threshold made from test partition

Table 4.1: Per-Class area thresholds expressed in percentage terms

Class	**Training Bound (%)**		**Test Bound (%)**	
	Min	Max	Min	Max
road	0.06	58.30	6.79	57.24
sidewalk	0.00	38.47	0.00	29.67
building	0.00	74.00	0.00	85.05
wall	0.00	42.87	0.00	14.03
fence	0.00	31.81	0.00	18.18
pole	0.00	6.94	0.00	5.11
traffic light	0.00	2.02	0.00	1.84
traffic sign	0.00	6.48	0.00	5.03
vegetation	0.00	59.14	0.00	60.39
terrain	0.00	26.54	0.00	18.13
sky	0.00	25.15	0.00	22.92
person	0.00	35.22	0.00	22.58
rider	0.00	3.65	0.00	5.08
car	0.00	40.08	0.00	48.18
truck	0.00	33.48	0.00	23.62
bus	0.00	26.91	0.00	47.70
train	0.00	60.72	0.00	43.34
motorcycle	0.00	9.85	0.00	2.66
bicycle	0.00	13.78	0.00	9.15

Table 4.2: Accuracy and costs of the usage of the training and test partition for threshold computation in the Area Unit (The first column states for the dataset partition employed for boundaries computation, the second states for accuracy obtained on the FOD dataset (corresponding CMs are shown in Figure 4.3 and 4.4). The last column states for COs classified as critical even if the mask is taken from the faultless network over a dataset partition

Boundaries Partition	**Accuracy**	**FP Cost**
Training	96.77%	0.46% on test set 0.2% on val set
Test	97.87%	2.08% on training set 2.6% on val set

Table 4.3: Accuracy achieved on the FOD and costs of the usage of the training and test partition for thresholds, when COs of a dataset partition are wrongly classified as critical for the Position Unit with the Incremental PWM

Boundaries Partition	**Accuracy**	**FP Cost**
Training	96.24%	0.46% on test set 0.2% on val set
Test	98.00%	2.22% on training set 1.8% on val set

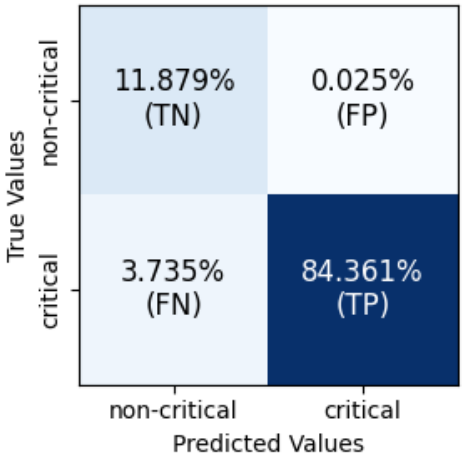


Figure 4.5: CM of the position unit with threshold made from training partition using the incremental PWM

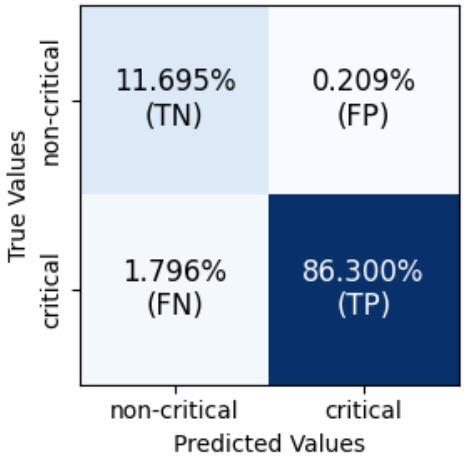


Figure 4.6: CM of the position unit with threshold made from test partition using the incremental PWM

As for the area unit, it is convenient to use the training partition for threshold computation, even if the corresponding accuracy is inferior.

The union of the two methods allows to obtain an improvement in the accuracy, keeping FP low and reducing the number of FN. CM with contributes of area and position unit with the chosen setup is shown in Figure 4.7.

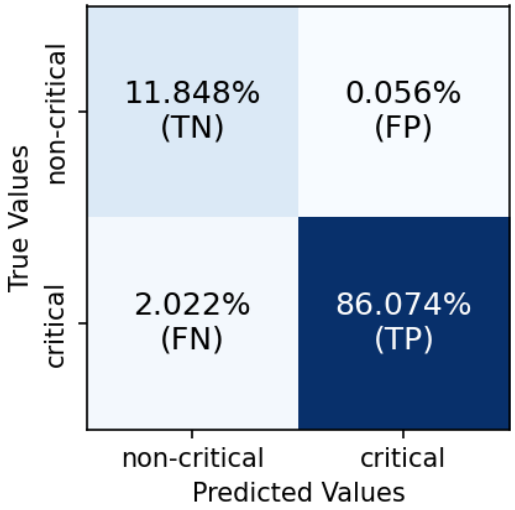


Figure 4.7: CM of area and position metrics with incremental PWM and boundaries from training set (accuracy of 97.92 %).

To take into account probabilities to find or not a class in a specific pixel, it is possible to substitute the incremental PWM, with probabilistic per-class PWMs. Probabilities are collected from training and test partitions and for this step, different PWMs are taken into account: the first ones, namely (p)-PWMs, are made of 19 matrices where each value is associated to the probability that a pixel is labeled with that class, while the other one, $(1-p)$-PWMs, computed by subtracting (p)-PWMs to 1, take into account the probability to **not** find that class in that specific pixel. Performances achieved with the two possible configurations are shown in Table 4.4 and the corresponding CMs are shown respectively in Figures 4.8, 4.9, 4.10 and 4.11.

Using the probabilities $(1-p)$ instead of (p) is much more functional, since a pixel labeled with an unlikely class has an impact on the metric result, while in the other case, considering a critical mask with classes located in both probable and non-probable pixels, the contribution of the latter is completely canceled. Combining each method with the area metric does not allow to improve the accuracy on the FOD dataset, providing even higher costs on COs. In addition,

Table 4.4: Accuracy achieved on the FOD and costs of the usage of the training and test partition for thresholds, when COs of a dataset partition are wrongly classified as critical in the position unit with probabilistic PWMs taken from a different partition for the Position Unit

Probabilistic-PWMs	**Boundaries Partition**	**Accuracy**	**Cost**
(p) from Training	Test	89.93%	2.40% on val set
(p) from Test	Training	83.49%	0.80% on val set
(1 - p) from Training	Test	98.17%	2.60% on val set
(1 - p) from Test	Training	97.86%	0.10% on val set

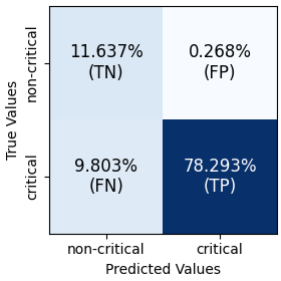


Figure 4.8: CM of position metric with probabilistic PWM (probabilities p from training set) and boundaries from test set

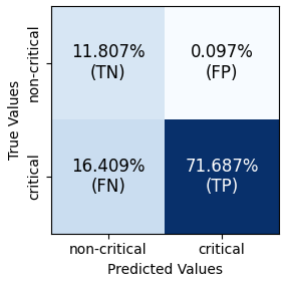


Figure 4.9: CM of position metric with probabilistic PWM (probabilities p from test set) and boundaries from training set

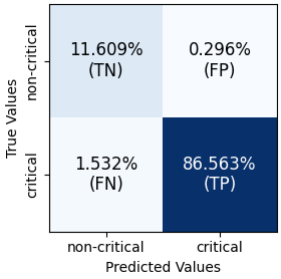


Figure 4.10: CM of position metric with probabilistic PWM (probabilities $1 - p$ from training set) and boundaries from test set

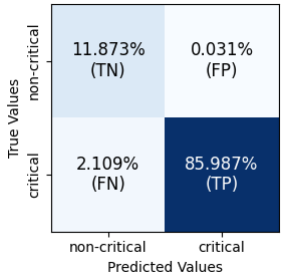


Figure 4.11: CM of position metric with probabilistic PWM (probabilities $1 - p$ from test set) and boundaries from training set

probabilities computation requires an additional overhead, therefore these methods reveal inefficient in this case.

4.1.3 Symmetries

Regarding the symmetry unit, the computation of boundaries over training and test partition allows to reach higher values of accuracy, as shown in Table 4.5.

Table 4.5: Accuracy achieved on the FOD and costs of the usage of the training and test partition for thresholds, when COs of a dataset partition are wrongly classified as critical in the Symmetry Metric with different SWMs. The (p) label is used to identify the SWMs that uses the probability that a pixel is vertically or horizontally symmetric. The $(1-p)$ label is used to identify the SWMs that uses the probability that a pixel is NOT vertically or horizontally symmetric

SWM	**Boundaries Partition**	**Accuracy**	**FP Cost**
Incremental	Training	97.87%	0.66% on test set 1.00% on val set
Incremental	Test	98.72%	0.66% on tr. set 3.80% on val set
(p) from Training	Test	89.37%	2.00% on val set
(p) from Test	Training	80.34%	1.00% on val set
$(1-p)$ from Training	Test	98.03%	3.00% on val set
$(1-p)$ from Test	Training	97.16%	1.20% on val set

As for the position unit, the incremental SWM with boundaries made on the training partition outperforms all the other methods.

Combining the three most basic setups seen until now, with boundaries made on the training set for all the metrics and the usage of incremental PWM and SWM, the accuracy reaches the 98.64% with a cost of 1.05 % and 1.40 % respectively on test and validation set.

4.1.4 Right Angles

The final component to be set up is the right angles unit, which, like the initial ones, requires calculating the maximum and minimum thresholds, basing on the parameter n (number of pixels aligned, described in chapter 3).

As shown in Table 4.6, accuracy reached with $n = 3$ is $\approx 45\%$ higher than with $n = 5, 7$. The origin of this discrepancy relies on the fact that lower values of n allows to capture a higher number of right angles: analyzing the boundaries, with $n = 5, 7$ the lower bound is 0, while with $n = 3$, the minimum is different from 0

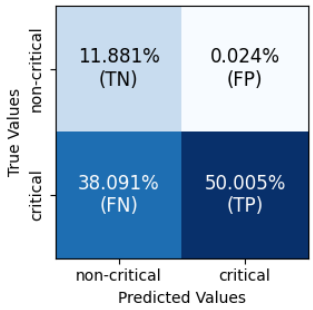


Figure 4.12: CM of the right angles unit with thresholds from training set and $n = 3$

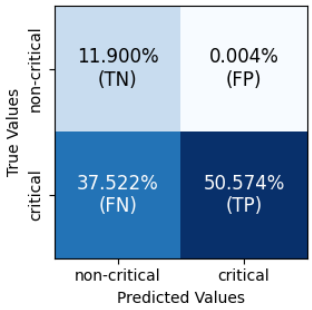


Figure 4.13: CM of the right angles unit with thresholds from test set and $n = 3$

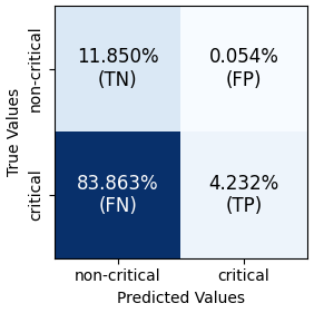


Figure 4.14: CM of the right angles unit with thresholds from training set and $n = 5$

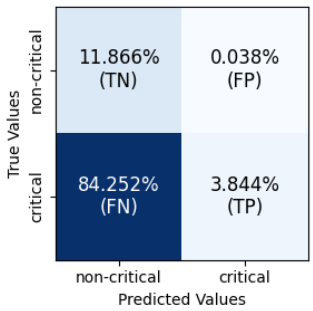


Figure 4.15: CM of the right angles unit with thresholds from test set and $n = 5$

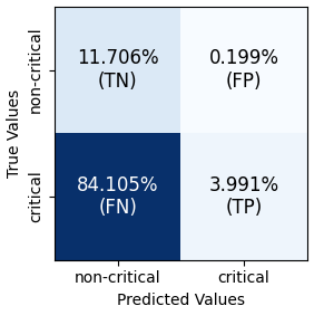


Figure 4.16: CM of the right angles unit with thresholds from training set and $n = 7$

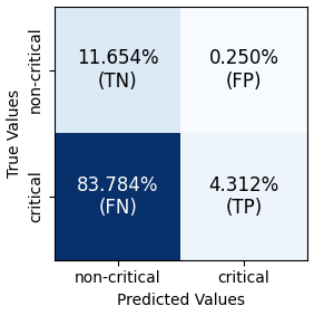


Figure 4.17: CM of the right angles unit with thresholds from test set and $n = 7$

Table 4.6: Performances on FOD and and costs of the usage of the training and test partition for thresholds, when COs of a dataset partition are wrongly classified as critical with the usage of different values of n aligned pixels for the Right Angles Unit

n	Partition	Boundaries	Accuracy	FP Cost
3	Training	(13, 1220)	61.89%	0.07% on test set 0.20% on val set
	Test	(34, 1446)	62.47%	0.27% on training set 0.00% on val set
5	Training	(0, 118)	16.08%	0.07% on test set 0.00% on val set
	Test	(0, 132)	15.71%	0.00% on training set 0.00% on val set
7	Training	(0, 32)	15.70%	0.00% on training set 0.00% on val set
	Test	(0, 29)	15.97%	0.10% on training set 0.20% on val set

Table 4.7: Performances on FOD and costs of the usage of the training partition for thresholds, when COs of a dataset partition are wrongly classified as critical with the usage of all the metrics combined with different values of n aligned pixels for the Right Angles Unit

n	Accuracy	FP Cost
3	98.95%	1.11% on test set 1.60% on val set
5	99.12%	1.11% on test set 1.40% on val set
7	99.00%	1.05% on test set 1.40% on val set

for both the partitions. Therefore, all those masks where the fault causes the effect of having the same class for all pixels (as seen in the chapter 3, they represent the 43.07 % of the FOD), are not labeled as critical, since having a value of right angles up to 0, this value falls within the limits. Nonetheless, considering the fact that the objective of this metric is not to identify that kind of masks and every metric has an impact on detection of different anomalies, Table 4.7 shows results reached by adding this unit with the previous set up, using different values of n as before. For consistency with the other metrics, only the training partition is adopted for thresholds of this metric. Results reveal that the usage of $n = 5$ allows to improve the performances in terms of accuracy and costs.

4.1.5 Final Set Up

Since FOD must not be used for any parameter selection, the proposed final metric is characterized by boundaries from training set for each unit, incremental weight matrices for position and symmetry and right angles with $n = 3$, which are the simplest and less expensive set ups, with statistics made on COs belonging to the same partition employed to train the network and most likely similar to the ground truth (consider that the ground truth of the test set is not available, differently from training and validation partition). Final results are shown in Figure 4.18.

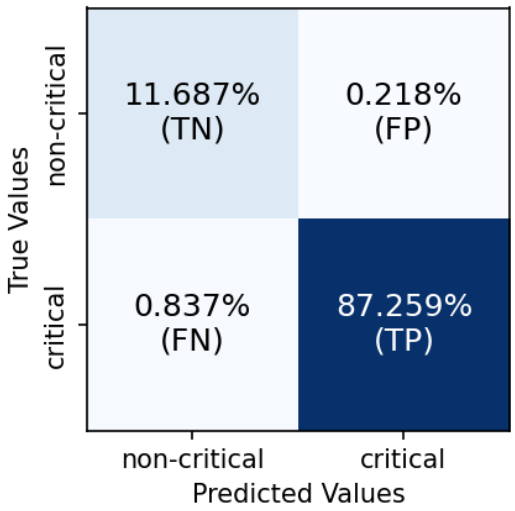


Figure 4.18: CM of the final method (accuracy of 98.95 %) using widest partition for threshold computation (training set), the cheapest weight matrices (incremental weight matrices) and the lowest possible n aligned pixel for the right angles unit (n=3)

4.2 Units Combinations

As soon as the setup is complete, method combinations are exploited in order to identify the units that allow obtaining greater accuracy and those that instead worsen results. The complete combinations study with the final setup is shown in Table 4.8.

Table 4.8: Performances on FOD and and costs of each metric combination (with the specific setup proposed in the previous section), when COs of a dataset partition are wrongly classified as critical.

Combination	**Accuracy**	**Cost**
('area', 'pos')	97.92 %	0.59 % on test set 0.40 % on val set
('area', 'symm')	98.56 %	0.92 % on test set 1.20 % on val set
('area', 'ra3')	97.44 %	0.52 % on test set 0.40 % on val set
('pos', 'symm')	98.44 %	0.92 % on test set 1.20 % on val set
('pos', 'ra3')	97.00 %	0.52 % on test set 0.40 % on val set
('symm', 'ra3')	98.34 %	0.72 % on test set 1.20 % on val set
('area', 'pos', 'symm')	98.64 %	1.05 % on test set 1.40 % on val set
('area', 'pos', 'ra3')	98.42 %	0.66 % on test set 0.60 % on val set
('area', 'symm', 'ra3')	98.89 %	0.98 % on test set 1.40 % on val set
('pos', 'symm', 'ra3')	98.81 %	0.98 % on test set 1.40 % on val set
('area', 'pos', 'symm', 'ra3')	98.95 %	1.11 % on test set 1.60 % on val set

Results show that the combination of all the units is more suitable in terms of accuracy. In particular, as long as metrics are combined, the accuracy generally increases, implying that each metric contributes in the identification of critical faults.

4.3 Computational Costs

The computational cost due to the use of this metric is evaluated at different levels:

- Single metric
- Final metric
- Statistics computation
  - With incremental weight matrices
  - With probabilistic weight matrices
- Final decision

Regarding the single metrics, computation of the area costs:

$$T_A(c, H, W) = O(c \times H \times W) \tag{4.1}$$

where $c = 19$ (number of classes), $H = 1024$ and $W = 2048$ (dimensions of the final mask) and it does not involve any floating point operation;
the position metric incurs a cost:

$$T_P(c, H, W) = 2 \times O(c \times H \times W) = O(c \times H \times W) \tag{4.2}$$

since for each class an element-by-element product is performed and the final value is given by the summation of all the values. Depending on the position weight matrix, if the incremental one is employed, no floating point operations are required; with the probabilistic ones, the number of floating point operations is 79691776;
the computation of the symmetry unit cost is:

$$\begin{aligned} T_S(c, H, W) &= 4 \times O\left(c \times H \times \frac{W}{2}\right) + 4 \times O\left(c \times \frac{H}{2} \times W\right) = \\ &= O\left(c \times H \times W\right) \end{aligned} \tag{4.3}$$

one contribute is given by the flipping operation, one by the logical-and between the half masks, the other one for the products and the last one for the whole summation. The same cost is required for both horizontal and vertical symmetry. Also here, depending if probabilistic weight matrices are used, the floating point operations are given by the product and the summation of the values. Considering both horizontal and vertical symmetries with probabilistic weight matrices, the total number of floating point operation required is 79691776 (the same of the position metric);

the right angles unit cost is:

$$T_{RA} = O(H \times W) + 2 \times O(H \times W) = \\ = O(H \times W) \tag{4.4}$$

given by the perimeter computation and the right angles count. This unit does not involve any floating point operation.

The cost of the final metric considering all the units is

$$T_{FM}(c, H, W) = T_A + T_P + T_S + T_{RA} = \\ = O(c \times H \times W) \tag{4.5}$$

and no floating point operations are needed with incremental weight matrices. The usage of the probabilistic ones requires a cost of 159383552 floating point operations, however low, compared to the operations that a deep neural network trained to detect the fault would require.

In total, considering all the metrics, they provide $4 \times c + 1$ values, therefore, the cost to check if each value is contained in the boundaries is:

$$T_C(c) = 2 \times O(4 \times c + 1) = O(c) \tag{4.6}$$

Regarding the statistics computation (SC) to define boundaries, the final cost is given by:

$$T_{SC_i}(N, c, H, W) = O(N \times c \times H \times W) + 2 \times O(N \times c) = \\ = O(N \times c \times H \times W) \tag{4.7}$$

where N is the size of the chosen dataset partition, which in this case is the training one ($N = 2975$) and i states for the usage of incremental weight matrices.

When probabilistic weight matrices are employed, an additional overhead of $O(M \times c \times H \times W)$ for the probabilities computation is required. Depending on the dataset partition adopted, $M = 1525$ for the test set or $N = 2975$ for the training one:

$$T_{SC_p}(c, H, W, N, M) = O(M \times c \times H \times W) + T_{SC_i} = \\ = O(M \times c \times H \times W) + O(N \times c \times H \times W) + 2 \times O(N \times c) = \\ = O\left((M + N) \times c \times H \times W\right) \tag{4.8}$$

The final decision (FD) is given by the computation of each metric and the comparison of each value with minimum and maximum values:

$$T_{FD} = T_{FM} + T_C = O(c \times H \times W) \tag{4.9}$$

4.4 Comparison and Safety Costs

Choosing to classify a mask as genuine when it is actually critically compromised by a fault (FN) has a different cost than classifying a genuine mask as critical (FP). In general, an autonomous driving system, in order to properly work, must minimize the risk of incorrectly identifying critical objects, as this could lead to fatal accidents. At the same time, it should avoid reporting genuine masks as critical with too high frequency, otherwise autonomous driving would not be possible.

This section presents an analysis of the various methods previously set up, each one compared basing on the normalized Detection Cost Function (DCF), which measures the risk of making a wrong decision, where the decision in this case is to classify the mask as critical or not. This measure is much more reliable than evaluating accuracy alone, especially in cases where costs and the dataset are unbalanced, allowing to take into account different costs and prior probabilities that the mask is critical or not.

The formula adopted for binary classification is the following:

$$\begin{aligned} \mathrm{DCF}_u(C_{fn}, C_{fp}, \pi_T) &= \pi_T C_{fn} P_{fn} + (1 - \pi_T) C_{fp} P_{fp} \\ \mathrm{DCF}(\pi_T, C_{fn}, C_{fp}) &= \frac{\mathrm{DCF}_u(\pi_T, C_{fn}, C_{fp})}{\min(\pi_T C_{fn}, (1 - \pi_T) C_{fp})} \end{aligned} \tag{4.10}$$

where DCF_u is the un-normalized DCF, also known as empirical Bayes risk, DCF is the normalized Bayes risk, the prior π_T, in this case, is the probability that a mask is critical with a stack-at-fault at bit 30, C_{fn} and C_{fp} are the costs for FNs and FPs and P_{fn} and P_{fp} are FN and FP rates, respectively computed as $FNR = \frac{FN}{FN+TP}$ and $FPR = \frac{FP}{FP+TN}$.

The first subsection provides a further comparison of each metric combination, from the point of view of the risk, while the second one compares the final set up with the state-of-the-art metrics. Each method is compared varying C_{fn} from 1 up to 100 and setting $C_{fp} = (100 - C_{fn}) + 1$. In addition, π_T is set with $\pi_{emp} = 0.88$, which is empirically computed as the frequency of critical faults in FOD.

4.4.1 Combinations Comparison

Combining each metric not only allows to obtain a higher accuracy, but also to decrease the Bayes risk. As shown in Figure 4.19, as each metric is added to the final evaluation, DCF values decrease. In particular, while in cases where the cost of misclassification is approximately the same, the difference is not particularly noticeable, in cases where C_{fn} is much higher than C_{fp}, the risk decreases significantly. However, as the C_{fn} increases with respect to the C_{fp}, the DCF increases significantly, with an almost exponential trend. The top four metric combinations

are shown in Figure 4.20. Note that `'area'`, `'pos'`, `'symm'` and `'ra3'` represent the methods used for the final metric (with bounds taken from training set and incremental weight matrices).

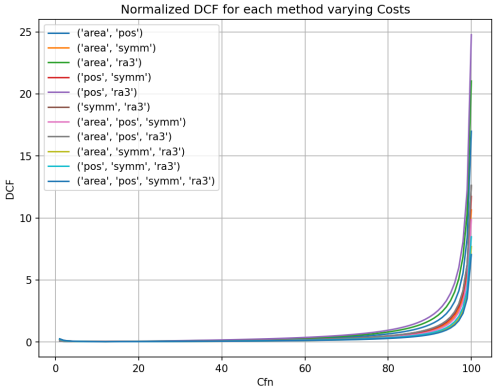


Figure 4.19: Normalized DCF of each metric combination varying C_{fn} (and C_{fp}) cost

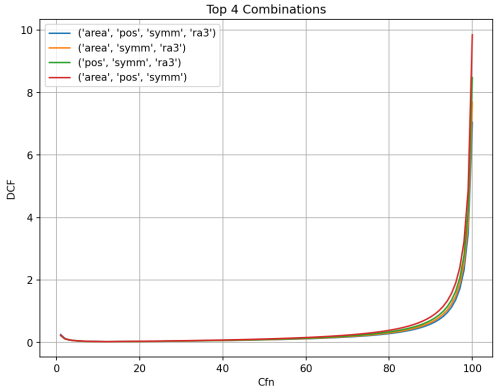


Figure 4.20: Top four metric combinations with respect to Normalized DCF varying C_{fn} (and C_{fp}) cost

4.4.2 State-of-the-art Comparison

Each method analyzed in this thesis is compared from different perspectives. At first, a summary comparison, presented in Table 4.9, evaluates them based on their requirements, highlighting how the developed metric, although an initial overhead, does not involve the ground truth or a fault-free network. Since the state-of-the-art methods classify the fault with more than two classes, these methods are adapted to the developed metric by defining different groupings, in order to make the multiclass classifications binary. CMs for each grouping are represented in Figures 4.21, 4.22 and 4.23 for the method relying on PA, while in Figures 4.24, 4.25 and 4.26 for the method with mIoU. The corresponding accuracy values are present in Table 4.10. Note that for greater visual clarity in graphs and tables, each class has been replaced with its initials as follows: `'c'` states for Critical, `'nc'` for Non-Critical, `'t'` for Tolerable, `'ni'` for No-impact SDC, `'w'` for Warning, `'a'` for Accepted and `'m'` for Masked.

As for the combinations, the overall risk of various methods is evaluated on FOD in term of normalized DCF.

Results show that even if the accuracy of the PA method is lower with respect to the mIoU, it reveals to be the safest when C_{fn} is particularly higher with respect to the C_{fp}. The developed metric (DM in Figures 4.27 and 4.28) is the riskiest, but results are still comparable with the state of the art and most importantly,

Table 4.9: Fault Detection methods requirements in comparison

Method	Requirements
Developed Metric	Statistics for thresholds computation, metrics extraction and comparison of each value with the corresponding threshold at test phase
PA Metric	CO at test phase and comparison with thresholds defined in [1]
mIoU Metric	Ground truth at test phase and comparison of mIoUs difference with thresholds defined in [2]

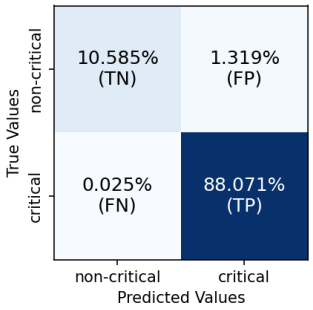


Figure 4.21: PA metric CM, made merging `'t'` and `'ni'` with `'m'` faults

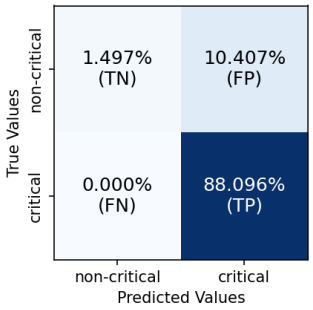


Figure 4.22: PA metric CM, made merging `'t'` with `'c'` faults and `'ni'` with `'m'` faults

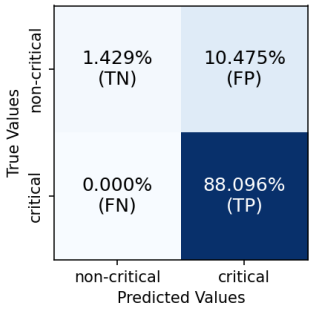


Figure 4.23: PA metric CM, made merging `'t'` and `'ni'` with `'c'` faults

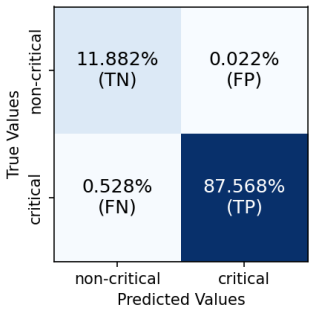


Figure 4.24: mIoU metric CM, made merging `'w'` and `'a'` with `'m'` faults

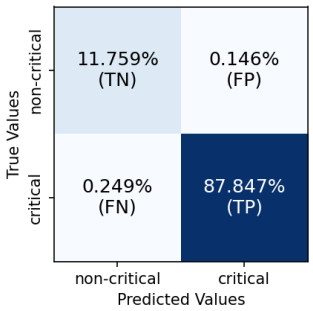


Figure 4.25: mIoU metric CM, made merging `'w'` with `'c'` faults and `'a'` with `'m'` faults

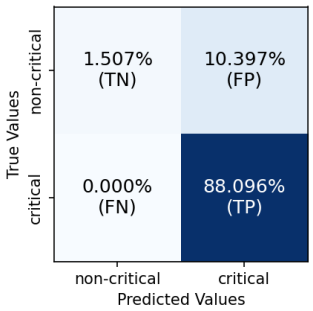


Figure 4.26: mIoU metric CM, made merging `'w'` and `'a'` with `'c'` faults

Table 4.10: Accuracy of different Fault Detection methods with different binary class aggregations

Method	**Accuracy**
Developed Metric (c vs nc)	98.95%
PA (c vs (t + ni + m))	98.66%
PA ((c + t) vs (ni + m))	89.59%
PA ((c + t + ni) vs m)	89.53%
mIoU (c vs (w + a + m))	99.45%
mIoU ((c + w) vs (a + m))	99.61%
mIoU ((c + w + a) vs m)	89.60%

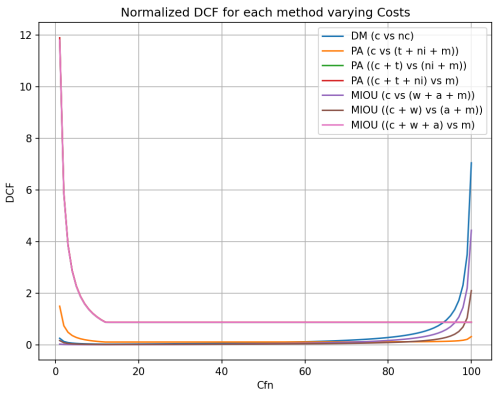


Figure 4.27: Normalized DCF of each method varying C_{fn} (and C_{fp}) cost

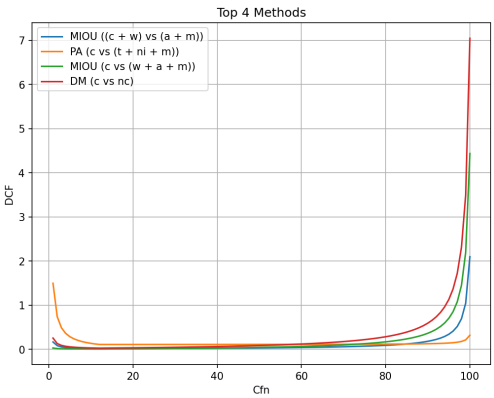


Figure 4.28: Top four methods with respect to Normalized DCF varying C_{fn} (and C_{fp}) cost

this metric does not use COs in the inference phase, but only beforehand for the statistical calculation of boundaries. The developed metric decision depends on thresholds, which can significantly improve or worsen the performance if varied. Tables 4.12, 4.11, 4.14, 4.13, 4.16, 4.15, 4.18, 4.17 show the wrong predictions for the top four safest methods (in terms of DCF) from Figure 4.28.

Table 4.11: Examples of masks wrongly classified as non-critical (FNs) by the developed metric, supplemented by mIoU and PA metrics

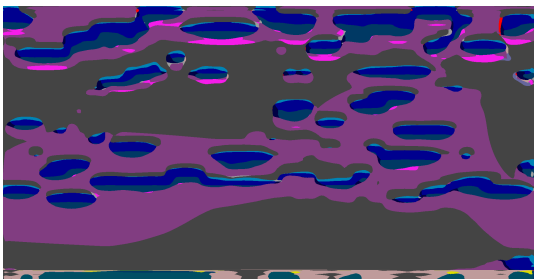
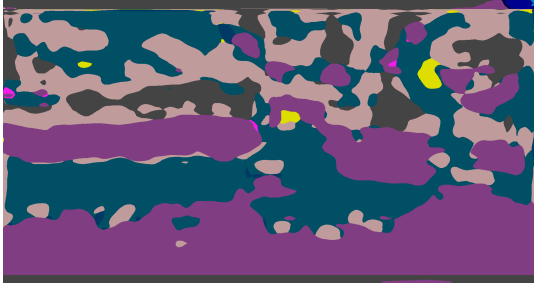
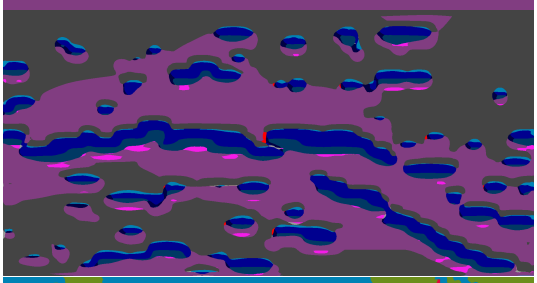
Frame	Injection	mIoU	PA	Result	Mask
455	13347	0.03	0.29	FN	
350	6093	0.04	0.29	FN	
444	13012	0.02	0.17	FN	
475	16013	0.82	0.99	FN	
136	12549	0.02	0.17	FN	

Table 4.12: Examples of masks wrongly classified as critical (FPs) by the developed metric, supplemented by mIoU and PA metrics

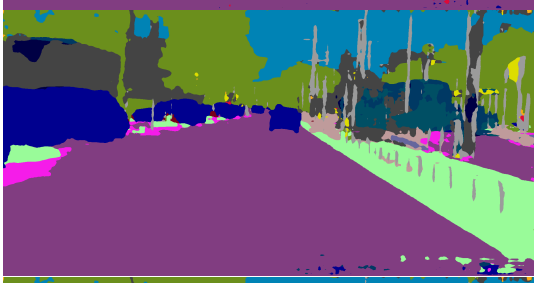
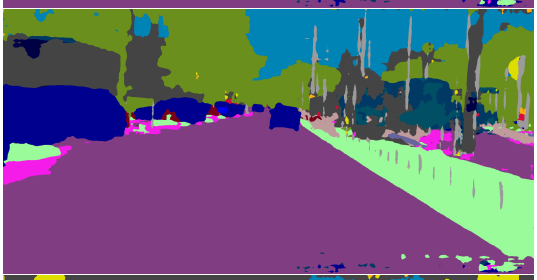
Frame	Injection	mIoU	PA	Result	Mask
49	15870	0.87	0.99	FP	
114	15870	0.93	0.99	FP	
9	15279	0.99	1.00	FP	
9	15223	0.66	0.96	FP	
265	10431	1.00	1.00	FP	

Table 4.13: Examples of masks wrongly classified as non-critical (FNs) by the mIoU metric, grouping Warning, Accepted and Masked faults into non-critical, supplemented by mIoU and PA metrics

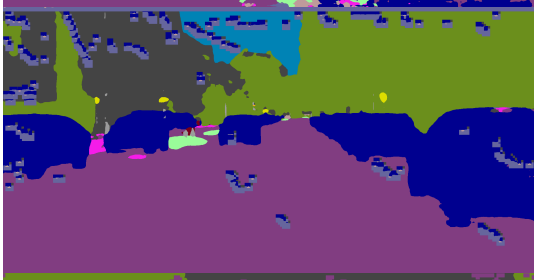
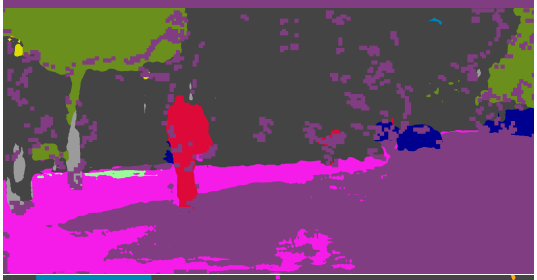
Frame	Injection	mIoU	PA	Result	Mask
22	15626	0.82	0.96	FN	
374	15934	0.70	0.93	FN	
460	16226	0.87	0.93	FN	
62	16311	0.92	0.98	FN	
283	15765	0.72	0.94	FN	

Table 4.14: Examples of masks wrongly classified as critical (FPs) by the mIoU metric, grouping Warning, Accepted and Masked faults into non-critical, supplemented by mIoU and PA metrics

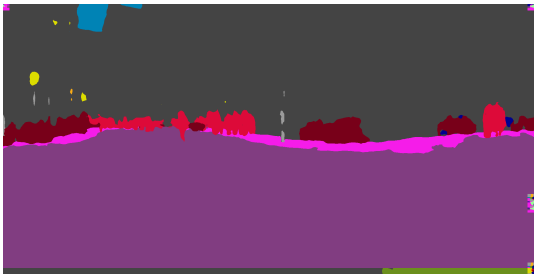
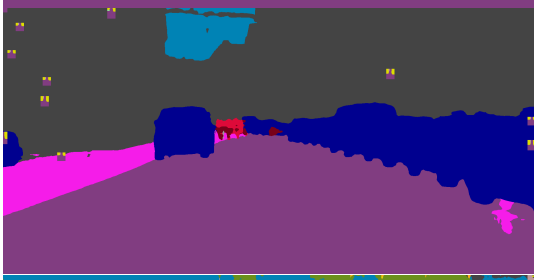
Frame	Injection	mIoU	PA	Result	Mask
306	15451	0.62	0.99	FP	
487	15479	0.63	0.99	FP	
370	15946	0.72	0.99	FP	
64	15290	0.67	0.98	FP	
282	15378	0.77	0.99	FP	

Table 4.15: Examples of masks wrongly classified as non-critical (FNs) by the mIoU metric, grouping Accepted and Masked faults into non-critical and Critical and Warning into critical, supplemented by mIoU and PA metrics

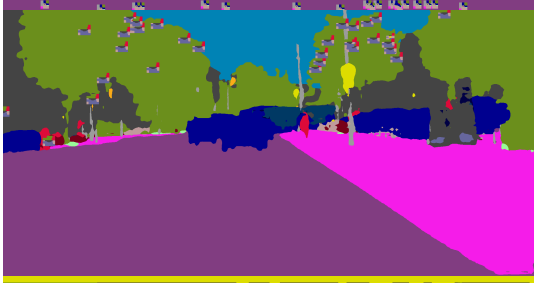
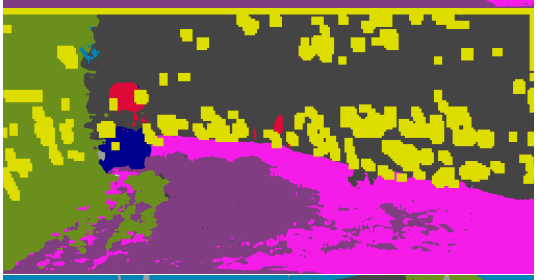
Frame	Injection	mIoU	PA	Result	Mask
402	15964	0.62	0.94	FN	
335	16013	0.86	0.98	FN	
494	15751	0.79	0.83	FN	
325	15765	0.75	0.96	FN	
108	15946	0.87	0.99	FN	

Table 4.16: Examples of masks wrongly classified as critical (FPs) by the mIoU metric, grouping Accepted and Masked faults into non-critical and Critical and Warning into critical, supplemented by mIoU and PA metrics

Frame	Injection	mIoU	PA	Result	Mask
370	15946	0.72	0.99	FN	
397	15634	0.60	0.97	FP	
289	15541	0.81	1.00	FP	
136	16276	0.64	0.93	FP	
275	15661	0.79	0.99	FP	

Table 4.17: Examples of masks wrongly classified as non-critical (FNs) by the PA metric, grouping Tolerable, No-Impact SDC and Masked faults into non-critical, supplemented by mIoU and PA metrics

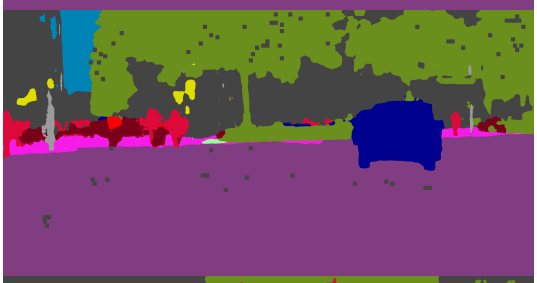
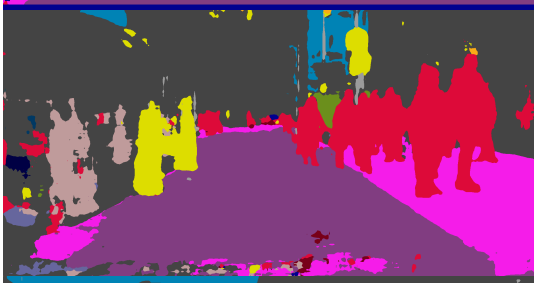
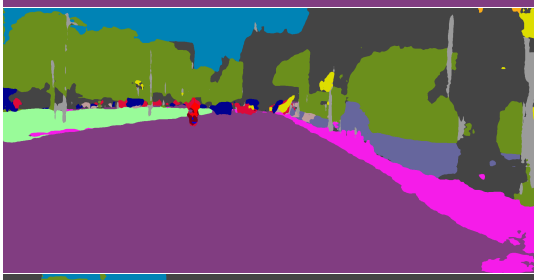
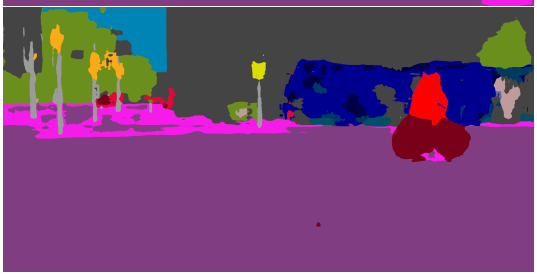
Frame	Injection	mIoU	PA	Result	Mask
18	16257	0.98	0.99	FN	
293	16257	0.98	0.99	FN	
273	16013	0.86	0.99	FN	
296	16257	0.99	1.00	FN	
183	15997	0.86	0.99	FN	

Table 4.18: Examples of masks wrongly classified as critical (FPs) by the PA metric, grouping Tolerable, No-Impact SDC and Masked faults into non-critical, supplemented by mIoU and PA metrics

Frame	Injection	mIoU	PA	Result	Mask
480	15654	0.79	0.97	FP	
95	450	0.80	0.99	FP	
383	2806	0.91	1.00	FP	
156	15317	0.88	0.99	FP	
61	15502	0.85	0.99	FP	

Chapter 5

Conclusions

Overall, this study explores different metrics to measure the quality of semantic segmentation masks. These metrics act measuring specific properties of a mask produced by the network. Comparing them with boundaries representing what is expected to be obtained from the network in the absence of faults it is possible to adopt them for fault detection purpose. While state-of-the-art methods require a comparison with the faultless network output or the ground truth to be computed, the proposed metrics imprint, within specific thresholds, the characteristic patterns of each class of the segmentation masks. However, unlike mIoU and PA, the method requires an initial overhead for the statistical calculation of unknown bounds, but at classification phase, the computational cost is equivalent to that of mIoU. The proposed metrics are tested on FOD, a dataset specifically developed within this thesis, also available for potential future research.

Results obtained on the FOD with the various combinations of metrics and the final one show that performances are very close to those of mIoU and PA, even if the number of FN obtained is considerably higher than the latter, which however, in cases where ground truth or CO is not available, cannot be used, not allowing to identify the fault.

The proposed approach can be extended using additional metrics that analyze appropriate characteristics of the segmentation masks. Furthermore, thresholds, which are currently linked to a single dataset, can be varied through different techniques that involve adapting the statistics obtained basing on the horizon line or generalizing from a single domain to another, in order to better describe the classes and allow the correct identification of the fault even when there is not enough data or in cases where the network is used in different domains than the one on which the network is trained or tested.

Bibliography

[1] Stéphane Burel, Adrian Evans, and Lorena Anghel. «Techniques for detecting and masking faults in semantic segmentation applications». In: *Microelectronics Reliability* 157 (2024), p. 115397. ISSN: 0026-2714. DOI: 10.1016/j.microrel.2024.115397. URL: https://www.sciencedirect.com/science/article/pii/S0026271424000775 (cit. on pp. 1, 14, 15, 17, 42).

[2] G. Govarini, A. Ruospo, and E. Sanchez. «A Fast Reliability Analysis of Image Segmentation Neural Networks Exploiting Statistical Fault Injections». In: *2023 IEEE 24th Latin American Test Symposium (LATS)*. 2023, pp. 1–6. DOI: 10.1109/LATS58125.2023.10154488 (cit. on pp. 1, 2, 11–14, 17, 42).

[3] Sparsh Mittal. «A survey on modeling and improving reliability of DNN algorithms and accelerators». In: *Journal of Systems Architecture* 104 (2020), p. 101689. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2019.101689. URL: https://www.sciencedirect.com/science/article/pii/S1383762119304965 (cit. on pp. 1, 7, 8, 12, 16, 20).

[4] Rudra PK Poudel, Stephan Liwicki, and Roberto Cipolla. «Fast-scnn: Fast semantic segmentation network». In: *arXiv preprint arXiv:1902.04502* (2019) (cit. on pp. 2, 5, 6).

[5] 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: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on pp. 2, 4).

[6] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. «Image Segmentation Using Deep Learning: A Survey». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.7 (2022), pp. 3523–3542. DOI: 10.1109/TPAMI.2021.3059968 (cit. on pp. 3, 4).

[7] Cesar Torres-Huitzil and Bernard Girau. «Fault and Error Tolerance in Neural Networks: A Review». In: *IEEE Access* 5 (2017), pp. 17322–17341. DOI: 10.1109/ACCESS.2017.2742698 (cit. on p. 7).

[8] J.A. Abraham and W.K. Fuchs. «Fault and error models for VLSI». In: *Proceedings of the IEEE* 74.5 (1986), pp. 639–654. DOI: 10.1109/PROC.1986.13528 (cit. on p. 7).

[9] A. Ruospo et al. «Assessing Convolutional Neural Networks Reliability through Statistical Fault Injections». In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2023, pp. 1–6. DOI: 10.23919/DATE56975.2023.10136998 (cit. on pp. 7, 11, 12).

[10] 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. 8–10).

[11] V. Turco, A. Ruospo, E. Sanchez, and M. Sonza Reorda. «Early Detection of Permanent Faults in DNNs Through the Application of Tensor-Related Metrics». In: *2024 27th International Symposium on Design & Diagnostics of Electronic Circuits & Systems (DDECS)*. 2024, pp. 13–18. DOI: 10.1109/DDECS60919.2024.10508918 (cit. on p. 12).

[12] Dongning Ma, Fred Lin, Alban Desmaison, Joel Coburn, Daniel Moore, Sriram Sankar, and Xun Jiao. «Dr. DNA: Combating Silent Data Corruptions in Deep Learning using Distribution of Neuron Activations». In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. ASPLOS '24. La Jolla, CA, USA: Association for Computing Machinery, 2024, pp. 239–252. ISBN: 9798400703867. DOI: 10.1145/3620666.3651349. URL: https://doi.org/10.1145/3620666.3651349 (cit. on p. 12).

[13] Le-Ha Hoang, Muhammad Abdullah Hanif, and Muhammad Shafique. «FT-ClipAct: Resilience Analysis of Deep Neural Networks and Improving their Fault Tolerance using Clipped Activation». In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2020, pp. 1241–1246. DOI: 10.23919/DATE48585.2020.9116571 (cit. on p. 12).

[14] Kuang-Hua Huang and Jacob A. Abraham. «Algorithm-Based Fault Tolerance for Matrix Operations». In: *IEEE Transactions on Computers* C-33.6 (1984), pp. 518–528. DOI: 10.1109/TC.1984.1676475 (cit. on p. 13).

[15] A.A. Al-Yamani, N. Oh, and E.J. McCluskey. «Performance evaluation of checksum-based ABFT». In: *Proceedings 2001 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. 2001, pp. 461–466. DOI: 10.1109/DFTVS.2001.966800 (cit. on p. 13).

[16] *SFIadvancedmodels*. https://https://github.com/cad-polito-it/SFIadvancedmodels (cit. on p. 26).