# Software-Only Semantic Diverse Redundancy for High-Integrity AI-Based Functionalities

Martí Caro, Axel Brando, Jaume Abella

# Software-Only Semantic Diverse Redundancy for High-Integrity AI-Based Functionalities

Martí Caro[†,‡], Axel Brando[†], Jaume Abella[†]
[†]Barcelona Supercomputing Center (BSC) Barcelona, Spain
[‡]Universitat Politecnica de Catalunya Barcelona, Spain

***Abstract*—Dual (DMR) and Triple Modular Redundancy (TMR), often with some form of diversity, are used in safety-critical systems to realize those functionalities at the highest integrity level providing fault detection and/or tolerance capabilities. Redundant executions are intended to provide bit-level identical results and, upon any mismatch, an error is assumed and recovery actions taken as needed.**

**In this paper, we note that many emerging AI-based functionalities are intrinsically stochastic (e.g., camera-based object detection), and hence, their correctness must be judged semantically, with room for variations across correct outcomes (e.g., confidence must be above a given threshold). Building on this observation, we propose strategies to create DMR and TMR implementations of AI-based functionalities that bring not only fault tolerance against random hardware faults, but also against AI model inaccuracies. Those strategies, which can be realized with software-only means and ported to virtually any computing platform, build on input data modifications affecting the inference computations, but not the expected semantic output (e.g., introducing some limited random noise in the input data).**

***Index Terms*—Diversity, Redundancy, Safety, DNN, AI**

## I. Introduction

Artificial Intelligence (AI) is increasingly used in safety-critical systems as an enabler for autonomous operation. For instance, camera-based object detection implemented with Deep Neural Networks (DNN) is used across a wide variety of applications and domains such as autonomous driving cars [1], rendezvous and docking operations in space [2], and in-cabin pilot monitoring in avionics [3] among others.

AI has been used so far mostly in fail-safe systems [4], where AI software has been generally relieved from inheriting safety requirements by incorporating appropriate non-AI monitors. However, autonomous operation, often related to fail-operational systems, does not generally allow keeping AI software without safety requirements. In fact, since AI software is in charge of high-integrity operations, such as steering and braking in automotive, or docking in space, integrity requirements can easily be the highest ones (e.g., DAL-A in avionics, ASIL-D in automotive). Therefore, it is common that the corresponding safety guidelines impose the use of diverse redundancy as a mandatory safety measure.

So far, diverse redundancy has been built with bit-level correctness in mind. For instance, lockstep processors use 2 or 3 identical cores running the same software with just some time staggering so that their internal state differs at any point in time and, upon a fault affecting all of them simultaneously, the errors generated (if any) are expected to differ and, at least, be detected. In this case, any discrepancy in the results is regarded as an error even if it is semantically innocuous for a specific application, since it is hard to tell whether it is or not a priori. However, in the context of AI software, such as DNNs, many applications do not require bit-level correctness and, instead, perform stochastic processes where bit-level different outcomes can be regarded as semantically identical. For instance, two object detectors identifying the same object class as the most likely one, with confidence values above a detection threshold, and with highly overlapping bounding boxes, can be regarded as providing identical outputs even if confidence values and bounding boxes differ.

This paper exploits this observation, i.e., *redundant AI software may afford bit-level differences across semantically identical outcomes*, to realize diverse redundancy in more efficient ways. In particular, we consider solutions where diverse redundancy can be used not only to deal with random hardware faults, but also to mitigate inaccuracies brought by AI models themselves (e.g., false positives and false negatives). Moreover, we do so containing system design and operation costs by preserving the original AI model. Otherwise, the cost of designing, training, and verifying two models could be prohibitive, as well as the cost of fetching twice the amount of AI model parameters, which may be in the order of several GBs in the case of some DNNs for camera-based object detection [5].

Our solution builds upon applying semantically-neutral transformations to the input data of the AI model (e.g., images for camera-based object detection) that alter inference computations so that random hardware faults are mitigated due to physical redundancy, but also small AI model inaccuracies can be mitigated due to input data diversity. In particular, we realize our transformations in the input images used by the You Only Look Once (YOLO) [6] object detector and show that a variety of image transformations yield comparable results to the original case, and they can be combined with different schemes to form a TMR system.

The rest of the paper is organized as follows. Section II provides some background and related work on diverse redundancy. Section III presents our proposal. Section IV evaluates it. Finally, Section V concludes this work.

## II. Background and Related Work

High-integrity systems and components require the use of some form of redundancy for, at least, error detection, and possibly correction. Data storage and communication often build on some form of error correction codes. For instance, single error correction double error detection (SECDED) codes for data storage are a popular solution for large memories, and can also be used to protect end-to-end communications by forwarding those codes along with the data.

However, combinatorial logic, and computing components in general, often need to resort to full redundancy. Dual (DMR) and Triple Modular Redundancy (TMR) are effective solutions for error detection and correction respectively if faults affect one of the redundant components only. However, some types of faults, such as those affecting clock signals and power

networks, can lead to simultaneous errors in all redundant components. Hence, some form of diversity is wanted across redundant instances to avoid scenarios where errors are identical and cannot be detected by comparing outputs.

The usual solution to realize DMR and TMR with diversity in CPUs consists of operating redundant cores with some time staggering so that, despite redundant cores are identical and execute identical software, their state at any time instant differs, and the simultaneous impact of a fault is expected to cause different errors – if any. The most popular case of such scheme is known as Dual Core LockStep (DCLS for short), and realized in some commercial microcontrollers, such as the Infineon AURIX processor family [7]. Similar schemes have been devised for GPUs [8], as well as software-only alternatives for cores [9]. However, all those solutions build upon bit-level accuracy so, while they could be applied directly for DNNs (either in CPU, GPU, or ad-hoc accelerators), they cannot be used to mitigate DNN model inaccuracies.

Some solutions exploit semantic redundancy for data varying little over time (e.g., camera-based object detection at high frames per second rates) [10], [11]. However, differently to our proposal, those solutions only provide error detection capabilities and can only be applied to problems with some (high) degree of timing redundancy. Our approach, instead, provides intrinsic error correction capabilities and poses no requirements on the timing relations across input data.

In our previous work [12], we presented TRUST, a scheme combining temporal redundancy, such as the previous works, as well as spatial redundancy to provide a DCLS-like solution, but with the secondary accelerator operating with lower-precision arithmetic to reduce complexity and power. While TRUST provides error correction capabilities, such as ITLS, it does not increase the model accuracy, and is still bound to problems where temporal redundancy exists. Moreover, ITLS could be realized with software support only, as opposed to TRUST.

Ensemble methods have previously been adopted to enhance the accuracy of DNNs [13], [14]. However, those works introduce substantial memory and computational overheads since weights change across models. To address this challenge, Gao et al. [15] introduced an ensemble approach that combines the outputs of three or four lower-complexity models (e.g., an ensemble of ResNet22, ResNet32, and ResNet44) to replace the original model (e.g., ResNet110). In their study, Gao et al. compare their ensemble technique with a Triple Modular Redundancy (TMR) method, and show that it incurs smaller overheads. However, their ensemble approach requires three different models with different weights, while we rely on the use of a single model. Therefore, their approach increases the cost of designing, training, and verifying several models which may be prohibitive. Furthermore, their evaluation focuses on object classification, which is an easier task than object detection.

Jon's et al. recent work [16] bears the closest resemblance to ours. Authors propose the use of image transformations to detect – but not correct – anomalous situations that could cause a potential faulty situation in an autonomous system. In particular, authors use a DNN to calculate the steering angle of a robot while driving between two lines, which is a much simpler task than the object detection task considered in our work. Their approach consists in executing the DNN with the original image and then performing a follow-up execution with a very slightly modified image, such that the steering angles calculated for both images are expected to be identical, i.e. within a calculated uncertainty threshold. However, in our work, we allow a higher degree of modification to be able to increase the overall system robustness through detecting a larger number of objects between the different components of the TMR system. We attempted to apply such approach to our problem, but the results were not promising, since the output of a DNN for object detection is composed of millions of values, and we found that the calculated uncertainty threshold, with the image modifications used in our work, is too large in fault-free cases, such that it cannot detect faulty operations.

## III. Software-Only Semantic Diverse Redundancy

Our approach builds on realizing a redundant scheme for the inference with a given AI model by altering the inputs across redundant instances in a way that outputs are expected to remain identical semantically speaking, and merging them conveniently, as illustrated in Figure 1. Note that to achieve maximum performance, a hardware solution comprising three parallel physical compute units must be made available. However, if developing a hardware solution is not an option, our solution could be implemented solely through software means. This can be achieved by executing the three input images in parallel on a single GPU, following the work from S. Alcaide et al. [17] which demonstrates how to implement a software-only TMR scheme using a single GPU.

For the sake of easing the explanation, we focus on a TMR scheme applied on the popular You Only Look Once v4 (YOLOv4) [6] object detection software for images, which is used in a plethora of applications, including commercial automotive systems, such as Apollo [18]. In particular, the task at hand is object detection, where, for each object present within an image, the model has to classify it (i.e., determine the class of the object, such as, a vehicle or a person), and determine its position and size by creating a bounding box (i.e., a rectangle surrounding the object).

### A. Efficient Diverse Redundancy

As shown in some recent work, DNNs such as the one in YOLOv4 may require fetching some GBs of weights to process each image [5], whereas images occupy up to few MBs. Hence, memory bandwidth is often saturated to fetch the DNN weights. Based on this fact, our redundancy scheme aims at preserving the same DNN model for all redundant inference processes with the aim of enabling appropriate execution approaches that allow sharing weights fetched across the redundant processes.

Therefore, if weights are fixed, the simplest way to introduce diversity with software-only means consists of altering the input data. Our goal is applying modifications to the input data so that it differs across the redundant processes, but they are semantically equivalent so that inference should lead to the same object detections. In particular, we have considered some of the image transformations provided by the CLoDSA open library [19], which would map to $F(x)$, $G(x)$ and $H(x)$ in the figure:

- Applying histogram equalization (EQ).
- Image sharpening (SH).
- Dropout (DR) by setting some pixels to zero.
- Applying Gamma correction (GC).
- Blurring the image applying a Gaussian filter (GB).

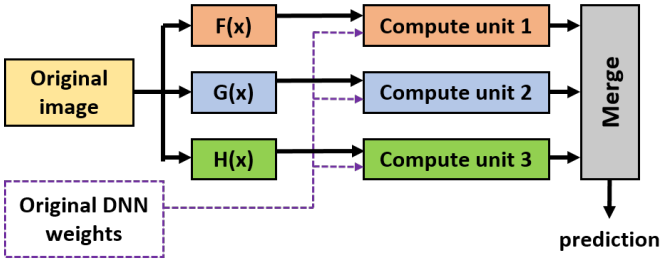


Fig. 1: Overview of our proposed TMR diverse redundancy scheme.

- Applying Gaussian noise (GN).
- Blurring the image applying the median filter (MB).
- Adding salt and pepper pixels (SP).
- Raising pixel values by a set amount (RV).
- Shifting the image some pixels to the right (RS).
- Shifting the image some pixels to the left (LS).
- Shifting the image some pixels to the top (TS).
- Shifting the image some pixels to the bottom (BS).
- Rotating the image clockwise at a certain angle (CR).
- Rotating the image anticlockwise at a certain angle (AR).
- Horizontally flipping the image (HF).
- Vertically flipping the image (VF).

To realize the TMR scheme, we select three different choices between the image transformations above and the original image, and perform inference for the three of them using the identical YOLOv4 DNN model. With this, we obtain three potentially different object detection outputs that need being combined properly.

### B. Combining Predictions

Rather than attempting to identify wrong predictions – if any – across the diverse and redundant predictions, we merge those predictions with the aim of making correct predictions supersede erroneous ones (see Figure 1). We build on the fact that predictions, even if correct, may not be identical at bit level. Hence, we opt for combining predictions whose detection class matches (e.g., a pedestrian is detected), and whose bounding box overlaps enough (e.g., bounding boxes have an intersection over union higher than 50%). However, it remains open how to set the final bounding box and confidence for the detection. Regarding the bounding box, note that model A can provide a bounding box overlapping above the threshold with the one from model B and the one from model C, but the ones from models B and C may overlap below the threshold. In this case, we still regard detections as the same detection and attempt to combine them.

**Confidence**. To set the confidence of the detection, we explore different schemes as follows:

- Voting: a detection is regarded as true if 2 out of 3 DNN models provide a confidence level above the detection threshold (e.g., using the default confidence threshold of 50% to determine that a detection should be correct).
- Averaging: a detection is regarded as true if the average of the confidence values for the three models is above the detection threshold.
- Maximum: a detection is regarded as true if at least one model indicates a confidence level above the detection threshold.

**Bounding box**. To set the bounding box of the detection, we perform it as follows:

- Voting: we retain the bounding box of the detection with the highest confidence among those being combined.
- Averaging: the bounding boxes of the 3 DNN models are averaged to obtain a final bounding box.
- Maximum: note that in this case, there may be overlapping bounding boxes that represent the same object. Therefore, a Non-Maximum Suppression (NMS) post-processing is applied to filter out repetitions. The NMS consists in filtering out the detections that represent the same object by comparing the bounding boxes of the detections of the same class, and if the bounding boxes have sufficient overlap (i.e., Intersection Over Union $> 0.5$), the bounding box with the highest confidence is kept, and the other overlapping bounding boxes are discarded.

Note that other approaches to determine the resulting bounding box are possible, such as selecting the largest bounding box among those being combined instead of considering the confidence level. However, using a bounding box not provided by any of the DNNs explicitly brings additional uncertainty. Hence, we stick to using a method that selects one of the relevant bounding boxes.

### C. Application to Safety-Relevant Systems

Deploying safety-relevant systems based on AI, with AI software inhering safety requirements, is still an open challenge [4]. Some relevant standards have seen the light very recently, such as ISO 5469 and ISO 21448 (aka SOTIF), but their practical application is not yet solved. Moreover, they do not provide specific guidance to tailor diverse redundancy. For instance, it is unclear how many levels and degrees of diverse redundancy are needed to make risk residual and system accuracy high enough. In our view, as part of the system architecture, diverse redundancy must exist at different levels, e.g., with multiple sensors, and with redundant and diverse processing for the data of each sensor. Our proposal aims at providing an efficient realization for the latter, hence enabling system designers with appropriate solutions to architect their AI-based safety-critical systems.

## IV. Evaluation

In this section, we introduce the evaluation setup (YOLO implementation, dataset, evaluation metrics, and fault injection framework), we discuss the results obtained with each image transformation when executing a single model, and we discuss the results obtained with our scheme, considering the different image transformations and results merging schemes in fault-free and faulty scenarios.

### A. Setup

We use a 32-bit floating point Tensorflow Keras implementation of YOLOv4 [20] with the publicly available pre-trained YOLOv4 parameters with the training subset of the Common Objects in Context (COCO) dataset [21]. We evaluate our proposal using the default 608x608 (image width x image height) network size of YOLOv4 and a reduced version using a 320x320 network size which roughly gives a 2x inference speed up at the cost of a slight accuracy drop [22], hence showing the effectiveness of our scheme regardless of the chosen network size. We evaluate our proposal with COCO [21],

using the validation subset, and keeping only those images that contain objects such as vehicles and pedestrians, which delves 870 images for evaluation.

However, the COCO dataset is not specific to AD, hence we have also evaluated our scheme with the KITTI dataset, designed specifically for AD applications. We have also considered using other AD datasets such as Udacity [23], Berkeley DeepDrive [24], and Waymo [25], but KITTI was the only one delivering relevant results for the pretrained YOLOv4 model due to using a comparable object labelling policies to the COCO dataset used for training. The KITTI dataset contains 7481 images captured from onboard vehicle cameras, and also includes three temporally preceding frames of each image, but captured at a very low FPS, which makes detections across images highly independent. Hence, we resorted to evaluating each image independently. The main limitation when using the KITTI dataset with the pretrained YOLOv4 model is that some classes (e.g., bus) are not labelled, and some classes (e.g., person, and motorbike) have significantly different labelling policies to those of COCO. Therefore, we have restricted our evaluation with the KITTI dataset to those classes with fewer discrepancies to obtain accurate results (i.e., car, van, and truck object classes). Another difference w.r.t. the COCO dataset is that the KITTI dataset includes some regions of the image – mostly background regions – that have not been labelled. Therefore, predictions made within these unlabelled zones are ignored during the evaluation process.

As evaluation metrics, we use the *Accuracy* (ACC) and *Mean Average Precision* (mAP). To obtain those, first we have to compute the *Intersection over Union* (IoU), i.e., the fraction of the intersection of the bounding boxes w.r.t. the union of those bounding boxes. In particular, we do so for the final objects detected by our proposal and the groundtruths. Since the detection process is stochastic and subject to some variation, whether an object is regarded as detected or not is done with a threshold, which in our case is 0.5, as set in other works [26]. This leads to true positives (TP) if the IoU is above the threshold, false positives (FP) if the IoU is below the threshold for a predicted object, and false negatives (FN) if the IoU of a groundtruth is below the threshold.

The accuracy (ACC) is obtained as follows:

$$ACC = \frac{TP}{TP + FP + FN} \tag{1}$$

The mAP is a more complex metric that is often the reference for object detection evaluation. The mAP leverages the TP, FP, and FN counts across the existing object classes to obtain an accuracy assessment for each object class, and then averages the individual accuracies to obtain an overall accuracy assessment. Given the complexity to introduce the details of this metric, and due to space constraints, we refer the interested reader to other publications for detailed explanations of this metric [27].

Regarding our fault injection campaign, we focus on random hardware faults impacting the computation (i.e., transient faults). We have injected transient faults in the result generated by multiplication or addition operations of the convolutional layers, as these layers account for over 99.5% of the total number of operations to process an image. Furthermore, random faults have only been considered to affect the sign or exponent of the floating-point number representation (i.e., 9 highest order bits), since random bit flips in the mantissa are mostly masked and do not lead to semantic changes in the outputs of the model [28], [29].

| Configuration | TP | FP | FN | ACC | mAP |
|---|---|---|---|---|---|
| BL | 3087 | 373 | 3104 | 47,03% | 48,69% |
| RV | 3085 | 372 | 3106 | 47,01% | 48,65% |
| GC | 3069 | 375 | 3122 | 46,74% | 48,38% |
| TS | 3061 | 353 | 3130 | 46,78% | 48,27% |
| HF | 3058 | 378 | 3133 | 46,55% | 48,16% |
| BS | 3050 | 358 | 3141 | 46,57% | 48,14% |
| LS | 3046 | 336 | 3145 | 46,67% | 48,09% |
| RS | 3037 | 379 | 3154 | 46,23% | 47,84% |
| GN | 3021 | 348 | 3170 | 46,20% | 47,67% |
| EQ | 3002 | 370 | 3189 | 45,76% | 47,38% |
| SP | 2991 | 341 | 3200 | 45,79% | 47,16% |
| AR | 2843 | 307 | 3348 | 43,75% | 44,87% |
| CR | 2834 | 292 | 3357 | 43,71% | 44,85% |
| GB | 2706 | 231 | 3485 | 42,14% | 42,97% |
| DR | 2659 | 287 | 3532 | 41,05% | 42,03% |
| SH | 2618 | 278 | 3573 | 40,47% | 41,32% |
| MB | 2550 | 239 | 3641 | 39,66% | 40,39% |
| VF | 317 | 36 | 5874 | 5,09% | 4,88% |

TABLE I: Results of each Image Transformation analysed independently with the COCO dataset (320x320 network size).

| Configuration | TP | FP | FN | ACC | mAP |
|---|---|---|---|---|---|
| EQ | 23023 | 3035 | 9727 | 64,34% | 68,27% |
| GC | 22737 | 2970 | 10013 | 63,65% | 67,37% |
| TS | 22424 | 2861 | 10326 | 62,97% | 66,53% |
| GB | 22323 | 2380 | 10427 | 63,54% | 66,50% |
| RV | 22375 | 2766 | 10375 | 63,00% | 66,42% |
| BS | 22301 | 2732 | 10449 | 62,85% | 66,19% |
| HF | 22298 | 2737 | 10452 | 62,83% | 66,12% |
| RS | 22267 | 2837 | 10483 | 62,57% | 66,10% |
| BL | 22247 | 2742 | 10503 | 62,68% | 66,07% |
| LS | 22229 | 2704 | 10521 | 62,70% | 65,97% |
| CR | 21464 | 2250 | 11286 | 61,33% | 63,93% |
| AR | 21425 | 2163 | 11325 | 61,37% | 63,88% |
| GN | 21338 | 2543 | 11412 | 60,46% | 63,34% |
| SP | 20805 | 2467 | 11945 | 59,08% | 61,71% |
| MB | 20556 | 2022 | 12194 | 59,12% | 61,32% |
| SH | 18424 | 2270 | 14326 | 52,61% | 54,54% |
| DR | 14882 | 1788 | 17868 | 43,09% | 44,00% |
| VF | 229 | 42 | 32521 | 0,70% | 0,62% |

TABLE II: Results of each Image Transformation analysed independently with the KITTI dataset (320x320 network size).

Tensorflow Keras poses difficulties to inject faults in the intermediate results of a layer, since layers operate as black boxes. To overcome this limitation, we have implemented a custom convolutional layer (same functionality but less efficient implementation). We choose an operation (addition or multiplication) randomly across all those operations in all convolutional layers. To inject a fault in a specific result of a specific layer, we execute the Keras model until the previous layer, replace the target layer with our custom one, inject the fault as needed (flipping a random bit in the sign or exponent), and resume the execution of the Keras implementation from the following layer onwards passing the result of our custom layer as needed.

### *B. Results of Independent Configurations*

Tables I, II, III, and IV show the TP, FP, and FN counts, as well as the ACC and mAP for each image transformation with

| Configuration | TP | FP | FN | ACC | mAP |
|---|---|---|---|---|---|
| TS | 3887 | 473 | 2304 | 58,33% | 61,12% |
| RV | 3868 | 467 | 2323 | 58,10% | 60,94% |
| HF | 3866 | 456 | 2325 | 58,16% | 60,88% |
| LS | 3856 | 435 | 2335 | 58,19% | 60,88% |
| BL | 3865 | 466 | 2326 | 58,06% | 60,87% |
| BS | 3860 | 439 | 2331 | 58,22% | 60,82% |
| RS | 3856 | 446 | 2335 | 58,10% | 60,76% |
| GC | 3859 | 470 | 2332 | 57,93% | 60,74% |
| GN | 3776 | 405 | 2415 | 57,25% | 59,61% |
| SP | 3708 | 398 | 2483 | 56,28% | 58,55% |
| EQ | 3699 | 450 | 2492 | 55,70% | 58,24% |
| CR | 3676 | 466 | 2515 | 55,22% | 57,83% |
| AR | 3675 | 504 | 2516 | 54,89% | 57,72% |
| SH | 3240 | 324 | 2951 | 49,73% | 51,15% |
| GB | 3213 | 267 | 2978 | 49,75% | 51,05% |
| MB | 2894 | 295 | 3297 | 44,62% | 45,71% |
| DR | 2335 | 298 | 3856 | 35,98% | 36,40% |
| VF | 366 | 34 | 5825 | 5,88% | 5,60% |

TABLE III: Results of each Image Transformation analysed independently with the COCO dataset (608x608 network size).

| Configuration | TP | FP | FN | ACC | mAP |
|---|---|---|---|---|---|
| EQ | 26193 | 4435 | 6557 | 70,44% | 77,59% |
| GC | 26093 | 4401 | 6657 | 70,23% | 77,14% |
| BS | 25930 | 4047 | 6820 | 70,47% | 76,86% |
| BL | 25935 | 4093 | 6815 | 70,39% | 76,86% |
| RV | 25949 | 4142 | 6801 | 70,34% | 76,85% |
| HF | 25894 | 3998 | 6856 | 70,46% | 76,75% |
| LS | 25890 | 4132 | 6860 | 70,20% | 76,73% |
| TS | 25822 | 4165 | 6928 | 69,95% | 76,50% |
| RS | 25811 | 4135 | 6939 | 69,98% | 76,46% |
| AR | 25607 | 3488 | 7143 | 70,66% | 76,03% |
| CR | 25533 | 3542 | 7217 | 70,35% | 75,81% |
| GN | 25548 | 3880 | 7202 | 69,75% | 75,64% |
| SP | 25229 | 3763 | 7521 | 69,10% | 74,65% |
| GB | 24796 | 2900 | 7954 | 69,55% | 73,93% |
| SH | 24765 | 3301 | 7985 | 68,69% | 73,21% |
| MB | 22545 | 2172 | 10205 | 64,56% | 67,36% |
| DR | 13191 | 1239 | 19559 | 38,81% | 38,83% |
| VF | 568 | 86 | 32182 | 1,73% | 1,56% |

TABLE IV: Results of each Image Transformation analysed independently with the KITTI dataset (608x608 network size).

the COCO and KITTI datasets using 320x320 and 608x608 network sizes.

First, note that the VF image transformation produces very low ACC and mAP results for all the datasets and network sizes evaluated, since the pretrained model has not been trained with vertically flipped objects. However, we keep this transformation for completeness.

In the case of COCO with 320x320 network size, the Baseline configuration provides the highest ACC and mAP, but with a network size of 608x608 there are 4 image transformations (TS, RV, HF, LS) providing up to 0.25% higher mAP than the baseline. For KITTI and 320x320 network size, there are 8 image transformations providing up to 2.2% higher mAP than the baseline, and with a 608x608 network size there are 3 image transformations (EQ, GC, BS) providing up to 0.73% higher mAP than the baseline.

Note that the pretrained YOLO model used COCO's training subset. Hence, the baseline model was expected to obtain high accuracy when evaluating similar images (e.g., COCO

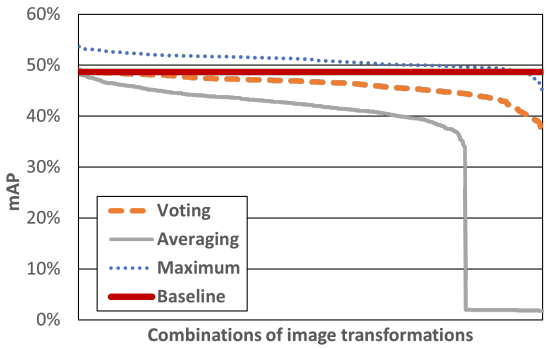


Fig. 2: Sorted mAP for all TMR F(x), G(X), and H(x) permutations for all merging algorithms using COCO (320x320 network size).

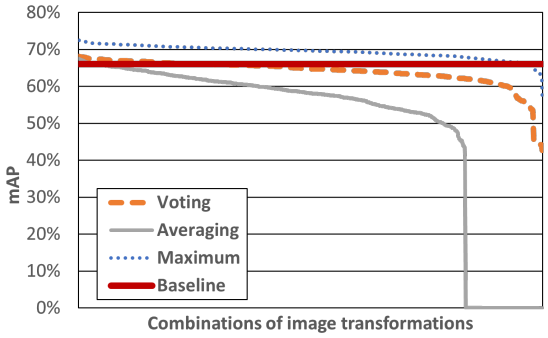


Fig. 3: Sorted mAP for all TMR F(x), G(X), and H(x) permutations for all merging algorithms using KITTI (320x320 network size).

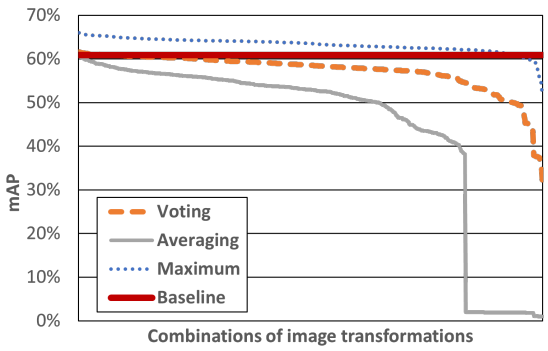


Fig. 4: Sorted mAP for all TMR F(x), G(X), and H(x) permutations for all merging algorithms using COCO (608x608 network size).

validation subset), but for different datasets, such as KITTI, it was indeed expected that some image transformations could surpass the accuracy of the baseline model.

### C. Results of the Merging Algorithms

We have obtained the TP, FP, and FN counts, as well as the ACC and mAP values for all TMR configurations and merging algorithms, namely Voting, Averaging and Maximum. Since the number of combinations is too large to allow reporting data for all those configurations (817 per merging algorithm), we show the mAP for all configurations in Figures 2, 3, 4,

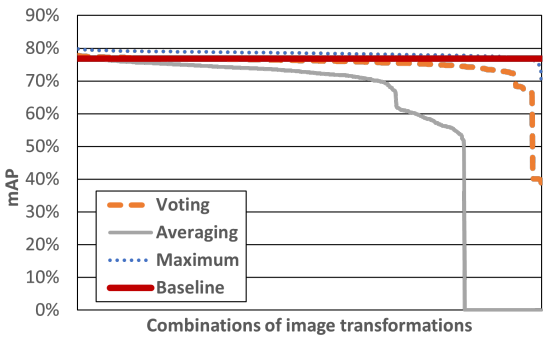


Fig. 5: Sorted mAP for all TMR F(x), G(X), and H(x) permutations for all merging algorithms using KITTI (608x608 network size).

| BASELINE | | | | | |
|---|---|---|---|---|---|
| **Configuration** | **TP** | **FP** | **FN** | **ACC** | **mAP** |
| **BL** | 3087 | 373 | 3104 | 47,0% | 48,69% |
| **VOTING** | | | | | |
| **Configuration** | **TP** | **FP** | **FN** | **ACC** | **mAP** |
| **BL, RV, BS** | 3098 | 357 | 3093 | 47,31% | 48,97% |
| **RV, LS, BS** | 3091 | 290 | 3100 | 47,69% | 48,95% |
| **BL, LS, BS** | 3087 | 296 | 3104 | 47,59% | 48,87% |
| **BL, RV, TS** | 3093 | 362 | 3098 | 47,20% | 48,85% |
| **GC, LS, BS** | 3083 | 283 | 3108 | 47,62% | 48,85% |
| **AVERAGING** | | | | | |
| **Configuration** | **TP** | **FP** | **FN** | **ACC** | **mAP** |
| **BL, BL, BL** | 3087 | 373 | 3104 | 47,03% | 48,69% |
| **BL, GC, RV** | 3079 | 370 | 3112 | 46,93% | 48,59% |
| **BL, GN, RV** | 3057 | 369 | 3134 | 46,60% | 48,24% |
| **GC, GN, RV** | 3052 | 371 | 3139 | 46,51% | 48,15% |
| **BL, GC, GN** | 3050 | 363 | 3141 | 46,54% | 48,15% |
| **MAXIMUM** | | | | | |
| **Configuration** | **TP** | **FP** | **FN** | **ACC** | **mAP** |
| **HF, RS, TS** | 3435 | 630 | 2756 | 50,36% | 53,65% |
| **HF, RS, BS** | 3432 | 645 | 2759 | 50,20% | 53,56% |
| **HF, LS, RS** | 3422 | 620 | 2769 | 50,24% | 53,48% |
| **HF, LS, BS** | 3418 | 616 | 2773 | 50,21% | 53,45% |
| **EQ, LS, TS** | 3409 | 606 | 2782 | 50,15% | 53,39% |

TABLE V: TMR results for the top-5 configurations of each merging algorithm using COCO (320x320 network size).

and 5, and the detailed results for the top-5 (in terms of mAP) for each algorithm in Tables V, VI, VII, and VIII.

Results in the figure are sorted independently for each merging algorithm, meaning that the $n^{th}$ best configuration for one algorithm may differ for the other algorithms despite being aligned w.r.t. the X-axis. Hence, the X-axis is not labelled. However, showing the data this way allows us to get several conclusions: (1) Maximum provides consistently the best results in terms of mAP across all merging algorithms; (2) Averaging is consistently worse, in terms of mAP, than the baseline when using the COCO dataset, but it is slightly better than the baseline in very few cases when evaluating the KITTI dataset; (3) Voting is slightly better than the baseline in some cases; and (4) all merging algorithms provide a smooth degradation of the mAP across image transformation combinations, hence meaning that, while some transformations may delve better results than others, there is not a strong dependence on very few combinations, so the approach provides robust

| BASELINE | | | | | |
|---|---|---|---|---|---|
| **Configuration** | **TP** | **FP** | **FN** | **ACC** | **mAP** |
| **BL** | 22247 | 2742 | 10503 | 62,68% | 66,07% |
| **VOTING** | | | | | |
| **Configuration** | **TP** | **FP** | **FN** | **ACC** | **mAP** |
| **EQ, GB, GC** | 22852 | 2614 | 9898 | 64,62% | 68,10% |
| **EQ, GC, LS** | 22867 | 2584 | 9883 | 64,72% | 68,03% |
| **EQ, GC, BS** | 22879 | 2642 | 9871 | 64,64% | 68,02% |
| **EQ, GC, TS** | 22875 | 2645 | 9875 | 64,63% | 68,02% |
| **HF, EQ, GC** | 22868 | 2651 | 9882 | 64,60% | 67,98% |
| **AVERAGING** | | | | | |
| **Configuration** | **TP** | **FP** | **FN** | **ACC** | **mAP** |
| **EQ, GB, GC** | 22700 | 2337 | 10050 | 64,70% | 67,54% |
| **EQ, GC, RV** | 22690 | 2704 | 10060 | 64,00% | 67,39% |
| **HF, EQ, GC** | 22636 | 2429 | 10114 | 64,35% | 67,33% |
| **BL, EQ, GC** | 22649 | 2667 | 10101 | 63,95% | 67,28% |
| **EQ, GB, RV** | 22569 | 2292 | 10181 | 64,41% | 67,18% |
| **MAXIMUM** | | | | | |
| **Configuration** | **TP** | **FP** | **FN** | **ACC** | **mAP** |
| **EQ, GB, RS** | 24495 | 4609 | 8255 | 65,57% | 72,53% |
| **EQ, RS, BS** | 24536 | 4808 | 8214 | 65,33% | 72,46% |
| **EQ, RS, TS** | 24517 | 4908 | 8233 | 65,10% | 72,44% |
| **EQ, GB, LS** | 24458 | 4503 | 8292 | 65,65% | 72,43% |
| **EQ, LS, BS** | 24506 | 4756 | 8244 | 65,34% | 72,39% |

TABLE VI: TMR results for the top-5 configurations of each merging algorithm using KITTI (320x320 network size).

| BASELINE | | | | | |
|---|---|---|---|---|---|
| **Configuration** | **TP** | **FP** | **FN** | **ACC** | **mAP** |
| **BL** | 3865 | 466 | 2326 | 58,06% | 60,87% |
| **VOTING** | | | | | |
| **Configuration** | **TP** | **FP** | **FN** | **ACC** | **mAP** |
| **HF, LS, TS** | 3901 | 403 | 2290 | 59,16% | 61,60% |
| **RV, LS, TS** | 3900 | 416 | 2291 | 59,03% | 61,54% |
| **GC, LS, TS** | 3897 | 410 | 2294 | 59,04% | 61,49% |
| **HF, RS, TS** | 3895 | 422 | 2296 | 58,90% | 61,48% |
| **HF, RV, BS** | 3893 | 417 | 2298 | 58,91% | 61,43% |
| **AVERAGING** | | | | | |
| **Configuration** | **TP** | **FP** | **FN** | **ACC** | **mAP** |
| **BL, BL, BL** | 3865 | 466 | 2326 | 58,06% | 60,87% |
| **BL, GC, RV** | 3862 | 460 | 2329 | 58,07% | 60,84% |
| **BL, HF, GC** | 3829 | 423 | 2362 | 57,89% | 60,42% |
| **BL, GN, RV** | 3827 | 425 | 2364 | 57,84% | 60,37% |
| **HF, GC, RV** | 3825 | 421 | 2366 | 57,85% | 60,35% |
| **MAXIMUM** | | | | | |
| **Configuration** | **TP** | **FP** | **FN** | **ACC** | **mAP** |
| **HF, RS, BS** | 4214 | 691 | 1977 | 61,23% | 66,01% |
| **HF, LS, TS** | 4207 | 713 | 1984 | 60,94% | 65,87% |
| **HF, LS, BS** | 4201 | 697 | 1990 | 60,99% | 65,84% |
| **HF, RS, TS** | 4204 | 715 | 1987 | 60,87% | 65,82% |
| **RS, LS, TS** | 4195 | 710 | 1996 | 60,79% | 65,72% |

TABLE VII: TMR results for the top-5 configurations of each merging algorithm using COCO (608x608 network size).

results.

Results in the tables offer a different angle to our analysis. For the COCO 320x320 configurations, in the case of Voting, we note that the bottom shift (BS), left shift (LS), and raise value (RV) transformations, as well as the baseline (BL), are the most popular ones since they appear in 4, 3, 3, and 3 of the top-5 TMR combinations respectively, and only Gamma correction (GC) appears once. In the case of Averaging, the best combination, despite not providing diversity, consists of

| BASELINE | | | | | |
|---|---|---|---|---|---|
| Configuration | TP | FP | FN | ACC | mAP |
| BL | 25935 | 4093 | 6815 | 70,39% | 76,86% |
| **VOTING** | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| HF, EQ, GC | 26199 | 4101 | 6551 | 71,09% | 77,76% |
| EQ, GC, LS | 26196 | 4105 | 6554 | 71,08% | 77,74% |
| EQ, GC, RS | 26184 | 4083 | 6566 | 71,09% | 77,70% |
| EQ, GC, BS | 26170 | 4122 | 6580 | 70,98% | 77,66% |
| EQ, LS, BS | 26118 | 3862 | 6632 | 71,34% | 77,61% |
| **AVERAGING** | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| HF, EQ, GC | 26057 | 3879 | 6693 | 71,14% | 77,32% |
| BL, HF, EQ | 26011 | 3748 | 6739 | 71,27% | 77,24% |
| EQ, GC, RV | 26064 | 4152 | 6686 | 70,63% | 77,21% |
| HF, EQ, RV | 26007 | 3781 | 6743 | 71,19% | 77,20% |
| BL, EQ, GC | 26052 | 4124 | 6698 | 70,65% | 77,20% |
| **MAXIMUM** | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| HF, EQ, LS | 26974 | 5877 | 5776 | 69,83% | 79,80% |
| HF, EQ, BS | 26963 | 5717 | 5787 | 70,09% | 79,79% |
| HF, EQ, TS | 26949 | 5772 | 5801 | 69,96% | 79,76% |
| EQ, LS, BS | 26947 | 5863 | 5803 | 69,79% | 79,73% |
| EQ, LS, TS | 26947 | 5901 | 5803 | 69,72% | 79,72% |

TABLE VIII: TMR results for the top-5 configurations of each merging algorithm using KITTI (608x608 network size).

using 3 times the BL. RV, GC, and Gaussian Noise (GN) are also quite frequent, and appear 3 times each one. Finally, in the case of Maximum merging, we observe that slightly shifting the original image in one direction is a frequent choice since 10 out of the 15 choices correspond to top, bottom, left or right shift. Hflip (HF) is also a frequent choice with 4 appearances, and Equalization (EQ) appears once.

For the COCO 608x608 configurations, in the case of Voting, we note that slightly shifting the original image in one direction is a frequent choice since 9 out of the 15 choices correspond to top, bottom, left or right shift, HF appears 3 times, RV appears twice, and GC appears once. In the case of Averaging, the best combination, despite not providing diversity, consists of using 3 times the BL. RV and GC appear three times, HF appears twice, and GN appears once. Finally, in the case of Maximum merging, we observe that slightly shifting the original image in one direction is a frequent choice since 11 out of the 15 choices correspond to top, bottom, left or right shift. HF is the second most frequent choice with 4 appearances.

For the KITTI 320x320 configurations, in the case of Voting, EQ and GC appear 5 times, shifting transformations appear 3 times, and HF appears once. In the case of Averaging the best configuration is no longer the baseline. EQ appears 5 times, GC appears 4 times, GB appears twice, and RV, HF, and BL appear once. Finally, in the case of Maximum merging, shifting appears 8 times, EQ 5 times, and GB appears twice.

For the KITTI 608x608 configurations, in the case of Voting, EQ and shifting appear 5 times each, GC appears 4 times, and HF appears once. In the case of Averaging, EQ appears 5 times, HF and GC appear 3 times, RV and BL appear twice. In the case of Averaging, EQ appears 5 times, and HF and GC appear 3 times. Finally, in the case of Maximum merging, shifting appears 7 times, EQ appears 5 times, and HF appears 3 times.

In summary, for the KITTI dataset, EQ is a particularly good option since it appears in all the TOP-5 configurations for all merging algorithms. The shifting image transformations works particularly well for both COCO and KITTI datasets when using a Voting or Maximum merging algorithm. The BL works particularly well for the COCO dataset when performing an Average merging, and the HF and GC are also noteworthy configurations for the KITTI dataset with an Average merging.

For the COCO dataset with both 320x320 and 608x608 network sizes, we note that Voting may increase a bit TPs and decrease a bit FPs. Hence, despite not providing the best mAP values, it provides an interesting tradeoff since it outperforms the baseline in all fronts. Averaging, instead, tends to decrease TPs while failing to decrease FPs sufficiently, and it is systematically worse than the baseline case. Finally, Maximum tends to increase TPs and FPs, which is expected since any object being detected by at least one of the three redundant inferences is regarded as a detection, and hence, a TP or a FP. Still, the combined effect clearly increases ACC and mAP values w.r.t. the baseline. Overall, looking at the results from the COCO dataset, if we care only about mAP or ACC, Maximum is clearly the best choice. If, instead, FPs are particularly problematic, Voting is the best solution.

For the KITTI dataset with both 320x320 and 608x608 network sizes, looking at the mAP values, Voting is clearly superior than Averaging for all configurations. However, looking at the ACC values, these approaches have closer results. Finally, Maximum tends to increase TPs and FPs, but the combined effect provides the highest mAP values. However, the ACC of the Maximum merging algorithm is slightly lower than the baseline for the KITTI dataset with a 608x608 network size. We ascribe this effect to the fact that the KITTI dataset does not properly label all background objects, which are mostly small. When increasing the network size, the model can detect more smaller objects, hence we observe a FP increase. We validated this observation in a subset of images by means of visual inspection, yet could not properly label the full dataset manually to fully fix this issue.

Overall, if we care only about mAP, Maximum merging is clearly the best choice for all the datasets analysed. If, instead, FPs are particularly problematic, Voting is the best solution.

### *D. Results of Fault Injection Configurations*

We have selected the top-mAP and top-ACC configurations for each merging algorithm, network size, and dataset, and analysed the impact of random fault injections on this subset of configurations. In the case of a single configuration providing both the top-mAP and top-ACC, this configuration with both the top-mAP and top-ACC, and the second highest mAP configuration have been selected instead.

First, we analyse the individual image transformations used in any of the top-mAP and top-ACC combinations indicated above. Tables IX, X, XI, and XII, present the fault-free results as well as the results after fault injection for those image transformations for the COCO 320x320, KITTI 320x320, COCO 608x608, and KITTI 608x608 setups, respectively. The tables provide the FP, FN, and TP count, as well as the ACC and mAP. Since some combinations (e.g., COCO 320x320 Averaging) use an ensemble of up to three baselines, we perform 3 different fault injections in such baseline, which we refer to as BL_1 AF, BL_2 AF, and BL_3 AF. For the

| BASELINE | | | | | |
|---|---|---|---|---|---|
| Configuration | TP | FP | FN | ACC | mAP |
| BL | 3087 | 373 | 3104 | 47,03% | 48,09% |
| INDEPENDENT CONFIGURATIONS | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| BL_1 AF | 2866 | 379 | 3325 | 43,62% | 44,57% |
| BL_2 AF | 2889 | 383 | 3302 | 43,95% | 44,75% |
| BL_3 AF | 2896 | 380 | 3295 | 44,07% | 45,02% |
| LS | 3046 | 336 | 3145 | 46,67% | 48,09% |
| LS AF | 2864 | 351 | 3327 | 43,78% | 44,25% |
| TS | 3061 | 353 | 3130 | 46,78% | 48,27% |
| TS AF | 2873 | 362 | 3318 | 43,84% | 44,53% |
| HF | 3058 | 378 | 3133 | 46,55% | 48,16% |
| HF AF | 2936 | 391 | 3255 | 44,61% | 45,87% |
| RV | 3085 | 372 | 3106 | 47,01% | 48,27% |
| RV AF | 2913 | 381 | 3278 | 44,32% | 45,36% |
| RS | 3037 | 379 | 3154 | 46,23% | 48,09% |
| RS AF | 2830 | 378 | 3361 | 43,08% | 44,03% |
| BS | 3050 | 358 | 3141 | 46,57% | 48,14% |
| BS AF | 2887 | 376 | 3304 | 43,96% | 44,73% |
| GC | 3069 | 375 | 3122 | 46,74% | 48,38% |
| GC AF | 2910 | 377 | 3281 | 44,31% | 45,43% |

TABLE IX: Results of the Fault Injection of the individual image transformations part of the combinations with the best mAP and ACC for the COCO dataset (320x320 network size).

| BASELINE | | | | | |
|---|---|---|---|---|---|
| Configuration | TP | FP | FN | ACC | mAP |
| BL | 3865 | 466 | 2326 | 58,06% | 60,87% |
| INDEPENDENT CONFIGURATIONS | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| BL_1 AF | 3667 | 487 | 2524 | 54,91% | 56,89% |
| BL_2 AF | 3661 | 463 | 2530 | 55,02% | 57,01% |
| BL_3 AF | 3633 | 468 | 2558 | 54,56% | 56,47% |
| HF | 3866 | 456 | 2325 | 58,16% | 60,88% |
| HF AF | 3708 | 450 | 2483 | 55,83% | 57,79% |
| TS | 3887 | 473 | 2304 | 58,33% | 61,12% |
| TS AF | 3703 | 471 | 2488 | 55,58% | 57,74% |
| BS | 3860 | 439 | 2331 | 58,22% | 60,82% |
| BS AF | 3681 | 408 | 2510 | 55,78% | 57,80% |
| RS | 3856 | 446 | 2335 | 58,10% | 60,76% |
| RS AF | 3653 | 439 | 2538 | 55,10% | 56,90% |
| LS | 3856 | 435 | 2335 | 58,19% | 60,88% |
| LS AF | 3667 | 418 | 2524 | 55,48% | 57,41% |
| RV | 3868 | 467 | 2323 | 58,10% | 60,94% |
| RV AF | 3664 | 440 | 2527 | 55,26% | 57,41% |
| GC | 3859 | 470 | 2332 | 57,93% | 60,74% |
| GC AF | 3710 | 461 | 2481 | 55,77% | 58,11% |

TABLE XI: Results of the Fault Injection of the individual image transformations part of the combinations with the best mAP and ACC for the COCO dataset (608x608 network size).

| BASELINE | | | | | |
|---|---|---|---|---|---|
| Configuration | TP | FP | FN | ACC | mAP |
| BL | 22247 | 2742 | 10503 | 62,68% | 66,07% |
| INDEPENDENT CONFIGURATIONS | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| EQ | 23023 | 3035 | 9727 | 64,34% | 68,27% |
| EQ AF | 21977 | 2945 | 10773 | 61,57% | 64,90% |
| LS | 22229 | 2704 | 10521 | 62,70% | 65,97% |
| LS AF | 21000 | 2606 | 11750 | 59,40% | 62,11% |
| GB | 22323 | 2380 | 10427 | 63,54% | 66,50% |
| GB AF | 21081 | 2324 | 11669 | 60,10% | 62,61% |
| GC | 22737 | 2970 | 10013 | 63,65% | 67,37% |
| GC AF | 21564 | 2897 | 11186 | 60,49% | 63,60% |
| RS | 22267 | 2837 | 10483 | 62,57% | 66,10% |
| RS AF | 21114 | 2742 | 11636 | 59,49% | 62,51% |
| BS | 22301 | 2732 | 10449 | 62,85% | 66,19% |
| BS AF | 21132 | 2616 | 11618 | 59,75% | 62,43% |
| RV | 22375 | 2766 | 10375 | 63,00% | 66,42% |
| RV AF | 21050 | 2693 | 11700 | 59,39% | 62,24% |

TABLE X: Results of the Fault Injection of the individual image transformations part of the combinations with the best mAP and ACC for the KITTI dataset (320x320 network size).

| BASELINE | | | | | |
|---|---|---|---|---|---|
| Configuration | TP | FP | FN | ACC | mAP |
| BL | 25935 | 4093 | 6815 | 70,39% | 76,86% |
| INDEPENDENT CONFIGURATIONS | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| HF | 25894 | 3998 | 6856 | 70,46% | 76,75% |
| HF AF | 24774 | 3832 | 7976 | 67,72% | 73,30% |
| EQ | 26193 | 4435 | 6557 | 70,44% | 77,59% |
| EQ AF | 24914 | 4218 | 7836 | 67,39% | 73,66% |
| GB | 24796 | 2900 | 7954 | 69,55% | 73,93% |
| GB AF | 23548 | 2767 | 9202 | 66,30% | 69,96% |
| AR | 25607 | 3488 | 7143 | 70,66% | 76,03% |
| AR AF | 24431 | 3348 | 8319 | 67,68% | 72,37% |
| GC | 26093 | 4401 | 6657 | 70,23% | 77,14% |
| GC AF | 24925 | 4221 | 7825 | 67,42% | 73,50% |
| LS | 25890 | 4132 | 6860 | 70,20% | 76,73% |
| LS AF | 24671 | 3998 | 8079 | 67,14% | 72,90% |
| CR | 25533 | 3542 | 7217 | 70,35% | 75,81% |
| CR AF | 24278 | 3399 | 8472 | 67,16% | 71,91% |

TABLE XII: Results of the Fault Injection of the individual image transformations part of the combinations with the best mAP and ACC for the KITTI dataset (608x608 network size).

remaining cases we report the fault-free (e.g., LS) and fault-injected cases (e.g., LS AF).

We observe that, in general, fault-injected configurations have lower TP counts, and except in COCO 320x320, also lower FP counts. ACC and mAP drop similarly for all configurations, with a 2.90% and 3.56% drop on average, respectively.

Tables XIII, XIV, XV, and XVI show the results of our proposal in the non-faulty TMR case, as well as in two faulty TMR cases; (i) with independents faults where for a given image only one of the three configurations is affected by a fault, and (ii) with faults in the same image for all three configurations. In the fault-free scenarios, which would correspond to virtually 100% of the time given that faults occur seldom, we obtain the following conclusion: (1) Voting and Maximum merging algorithms provide higher mAP and ACC than the Baseline, except for the HF, EQ, LS configuration for the KITTI dataset with a 608x608 network size, where we observe a slight ACC drop w.r.t. the baseline (for reasons previously discussed in Section IV-C), (2) Averaging only produces better results than the baseline with the KITTI dataset since the model was trained with the COCO training subset. Therefore, it is expected that some image transformations could surpass the accuracy of the baseline when using other datasets.

When faults occur independently, we obtain the following conclusions: (1) Voting provides higher mAP and ACC than the Baseline with both datasets and both network sizes

| BASELINE | | | | | |
|---|---|---|---|---|---|
| Configuration | TP | FP | FN | ACC | mAP |
| BL | 3087 | 373 | 3104 | 47,03% | 48,09% |
| VOTING | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| BL_1, RV, BS | 3098 | 357 | 3093 | 47,31% | 48,69% |
| BL_1, RV, BS (indep. faults) | 3071 | 356 | 3120 | 46,91% | 48,55% |
| BL_1, RV, BS (same frame) | 3029 | 343 | 3162 | 46,36% | 47,89% |
| RV, LS, BS | 3091 | 290 | 3100 | 47,69% | 48,66% |
| RV, LS, BS (indep. faults) | 3058 | 291 | 3133 | 47,18% | 48,39% |
| RV, LS, BS (same frame) | 3028 | 283 | 3163 | 46,77% | 47,92% |
| AVERAGING | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| BL_1, BL_2, BL_3 | 3087 | 373 | 3104 | 47,03% | 48,69% |
| BL_1, BL_2, BL_3 (indep. faults) | 2875 | 414 | 3316 | 43,53% | 45,13% |
| BL_1, BL_2, BL_3 (same frame) | 2594 | 447 | 3597 | 39,08% | 40,50% |
| BL_1, GC, RV | 3079 | 370 | 3112 | 46,93% | 48,66% |
| BL_1, GC, RV (indep. faults) | 2945 | 423 | 3246 | 44,53% | 46,31% |
| BL_1, GC, RV (same frame) | 2607 | 459 | 3584 | 39,20% | 40,73% |
| MAXIMUM | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| HF, RS, TS | 3435 | 630 | 2756 | 50,36% | 52,00% |
| HF, RS, TS (indep. faults) | 3415 | 761 | 2776 | 49,12% | 50,24% |
| HF, RS, TS (same frame) | 3393 | 1122 | 2798 | 46,40% | 44,17% |
| HF, RS, BS | 3432 | 645 | 2759 | 50,20% | 51,98% |
| HF, RS, BS (indep. faults) | 3412 | 788 | 2779 | 48,89% | 49,75% |
| HF, RS, BS (same frame) | 3398 | 1130 | 2793 | 46,41% | 44,26% |

TABLE XIII: TMR results for the best mAP and ACC configurations of each merging algorithm using the COCO dataset (320x320 network size).

| BASELINE | | | | | |
|---|---|---|---|---|---|
| Configuration | TP | FP | FN | ACC | mAP |
| BL | 3865 | 466 | 2326 | 58,06% | 60,87% |
| VOTING | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| HF, LS, TS | 3901 | 403 | 2290 | 59,16% | 61,60% |
| HF, LS, TS (indep.faults) | 3896 | 401 | 2295 | 59,10% | 61,51% |
| HF, LS, TS (same frame) | 3822 | 398 | 2369 | 58,01% | 60,36% |
| RV, LS, TS | 3900 | 416 | 2291 | 59,03% | 61,54% |
| RV, LS, TS (indep.faults) | 3880 | 418 | 2311 | 58,71% | 61,23% |
| RV, LS, TS (same frame) | 3831 | 410 | 2360 | 58,04% | 60,48% |
| AVERAGING | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| BL_1, BL_2, BL_3 | 3865 | 466 | 2326 | 58,06% | 60,87% |
| BL_1, BL_2, BL_3 (indep.faults) | 3697 | 490 | 2494 | 55,34% | 58,17% |
| BL_1, BL_2, BL_3 (same frame) | 3392 | 542 | 2799 | 50,38% | 53,15% |
| BL_1, GC, RV | 3862 | 460 | 2329 | 58,07% | 60,84% |
| BL_1, GC, RV (indep.faults) | 3751 | 472 | 2440 | 56,30% | 59,05% |
| BL_1, GC, RV (same frame) | 3449 | 510 | 2742 | 51,47% | 54,05% |
| MAXIMUM | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| HF, RS, BS | 4214 | 691 | 1977 | 61,23% | 66,01% |
| HF, RS, BS (indep.faults) | 4204 | 743 | 1987 | 60,63% | 64,32% |
| HF, RS, BS (same frame) | 4186 | 980 | 2005 | 58,37% | 58,75% |
| HF, LS, TS | 4207 | 713 | 1984 | 60,94% | 65,87% |
| HF, LS, TS (indep.faults) | 4201 | 823 | 1990 | 59,89% | 62,69% |
| HF, LS, TS (same frame) | 4188 | 1071 | 2003 | 57,67% | 57,59% |

TABLE XV: TMR results for the best mAP and ACC configurations of each merging algorithm using the COCO dataset (608x608 network size).

| BASELINE | | | | | |
|---|---|---|---|---|---|
| Configuration | TP | FP | FN | ACC | mAP |
| BL | 22247 | 2742 | 10503 | 62,68% | 66,07% |
| VOTING | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| EQ, GB, GC | 22852 | 2614 | 9898 | 64,62% | 68,10% |
| EQ, GB, GC (indep. faults) | 22796 | 2587 | 9954 | 64,51% | 67,95% |
| EQ, GB, GC (same image) | 22501 | 2516 | 10249 | 63,80% | 67,08% |
| EQ, GB, LS | 22760 | 2221 | 9990 | 65,08% | 67,94% |
| EQ, GB, LS (indep. faults) | 22668 | 2171 | 10082 | 64,91% | 67,67% |
| EQ, GB, LS (same image) | 22354 | 2104 | 10396 | 64,14% | 66,75% |
| AVERAGING | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| EQ, GB, GC | 22700 | 2337 | 10050 | 64,70% | 67,54% |
| EQ, GB, GC (indep. faults) | 21607 | 2375 | 11143 | 61,51% | 64,21% |
| EQ, GB, GC (same image) | 19734 | 2387 | 13016 | 56,16% | 58,56% |
| EQ, GC, RV | 22690 | 2704 | 10060 | 64,00% | 67,39% |
| EQ, GC, RV (indep. faults) | 21620 | 2702 | 11130 | 60,98% | 64,16% |
| EQ, GC, RV (same image) | 19707 | 2678 | 13043 | 55,63% | 58,37% |
| MAXIMUM | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| EQ, GB, RS | 24495 | 4609 | 8255 | 65,57% | 72,53% |
| EQ, GB, RS (indep. faults) | 24499 | 4845 | 8251 | 65,17% | 71,18% |
| EQ, GB, RS (same image) | 24442 | 5085 | 8308 | 64,60% | 69,49% |
| EQ, GB, BS | 24382 | 4226 | 8368 | 65,94% | 72,26% |
| EQ, GB, BS (indep. faults) | 24409 | 4440 | 8341 | 65,63% | 70,94% |
| EQ, GB, BS (same image) | 24330 | 4699 | 8420 | 64,97% | 69,18% |

TABLE XIV: TMR results for the best mAP and ACC configurations of each merging algorithm using the KITTI dataset (320x320 network size).

| BASELINE | | | | | |
|---|---|---|---|---|---|
| Configuration | TP | FP | FN | ACC | mAP |
| BL | 25935 | 4093 | 6815 | 70,39% | 76,86% |
| VOTING | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| HF, EQ, GC | 26199 | 4101 | 6551 | 71,09% | 77,76% |
| HF, EQ, GC (indep.faults) | 26168 | 4057 | 6582 | 71,10% | 77,68% |
| HF, EQ, GC (same frame) | 25910 | 3957 | 6840 | 70,59% | 76,92% |
| EQ, GB, LS | 25862 | 3426 | 6888 | 71,49% | 77,01% |
| EQ, GB, LS (indep.faults) | 25811 | 3387 | 6939 | 71,43% | 76,87% |
| EQ, GB, LS (same frame) | 25502 | 3293 | 7248 | 70,75% | 75,95% |
| AVERAGING | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| HF, EQ, GC | 26057 | 3879 | 6693 | 71,14% | 77,32% |
| HF, EQ, GC (indep.faults) | 25135 | 3846 | 7615 | 68,68% | 74,54% |
| HF, EQ, GC (same frame) | 23178 | 3776 | 9572 | 63,46% | 68,70% |
| HF, EQ, AR | 25853 | 3468 | 6897 | 71,38% | 76,85% |
| HF, EQ, AR (indep.faults) | 24946 | 3449 | 7804 | 68,91% | 74,13% |
| HF, EQ, AR (same frame) | 22901 | 3425 | 9849 | 63,31% | 67,99% |
| MAXIMUM | | | | | |
| Configuration | TP | FP | FN | ACC | mAP |
| HF, EQ, LS | 26974 | 5877 | 5776 | 69,83% | 79,80% |
| HF, EQ, LS (indep.faults) | 26956 | 5988 | 5794 | 69,59% | 78,79% |
| HF, EQ, LS (same frame) | 26921 | 6183 | 5829 | 69,15% | 77,33% |
| GB, CR, AR | 26182 | 4318 | 6568 | 70,63% | 77,79% |
| GB, CR, AR (indep.faults) | 26171 | 4449 | 6579 | 70,35% | 76,77% |
| GB, CR, AR (same frame) | 26142 | 4694 | 6608 | 69,82% | 75,31% |

TABLE XVI: TMR results for the best mAP and ACC configurations of each merging algorithm using the KITTI dataset (608x608 network size).

analysed. (2) Maximum provides higher mAP and ACC than the Baseline for the COCO dataset with both network sizes, and for the KITTI dataset with a 320x320 network size. However, for the KITTI dataset with a 608x608 network size, we observe that the HF, EQ, LS (indep. faults) configuration provides higher mAP but lower ACC (for reasons previously discussed in Section IV-C), and the GB, CR, AR (indep. faults) configuration provides higher ACC but lower mAP, since this configuration provides the highest ACC in fault-free cases, but does not deliver as high mAP as the top-mAP configuration. (3) Averaging produces worse results than the baseline with all configurations.

When faults occur in the same image for all three configurations, we obtain the following conclusions: (1) for the

COCO dataset, all merging algorithms provide lower accuracy than the baseline, (2) Voting produces the closest accuracy to the baseline, both in terms of ACC and mAP, (3) for the KITTI dataset, both Voting and Maximum have at least one configuration producing higher accuracy (either both ACC and mAP or only one of the metrics) than the baseline.

Overall, we can conclude the following (1) Averaging is consistently worse than Voting and Maximum in all cases, (2) Maximum produces the highest ACC and mAP results in the fault-free scenarios, as well as when faults occur independently, except for the KITTI dataset with a 608x608 network size, where the ACC of Voting is slightly higher, (3) Voting produces the highest mAP results with the COCO dataset when faults occur simultaneously, but regarding the ACC, two configurations produce higher ACC with Voting, while the other two produce higher ACC with Maximum. (4) When faults occur simultaneously for the KITTI dataset, Maximum provides higher ACC and mAP than Voting with a 320x320 network size, but Voting provides higher ACC with a 608x608 network size.

Overall, we can conclude that the Maximum merging algorithm produces the best mAP results, but it provides significantly higher FPs than Voting. Therefore, if FPs are a major concern, Voting is the best solution overall, while if the objective is to obtain the maximum mAP, the Maximum algorithm is the best solution since the fault-free scenario equals to virtually 100% of time, and typically faults will not affect all redundant components at the same time.

## V. Conclusions

AI software is increasingly used in safety-critical systems for functionalities where such software inherits safety requirements. Whenever those requirements relate to the highest integrity levels, AI software must be realized with diverse redundancy, such as, for instance, TMR. Existing solutions based on lockstep processors provide such diverse redundancy with bit-level error detection and correction. In this paper, we note that abundant AI functionalities do not need bit-level correctness and, instead, semantic correctness suffices due to the stochastic nature of the AI-based functionalities implemented. We leverage this observation to present a diverse redundancy scheme for AI models based on applying minor transformations to the input data with the aim of creating diverse, yet semantically identical, predictions, which we use to mitigate, apart from hardware random errors, also AI model errors.

As future work, we plan to extend our work evaluating the different image transformations and merging algorithms in DMR scenarios, which are particularly relevant for some domains, such as automotive.

## Acknowledgement

The research leading to these results has received funding from the European Union's Horizon Europe Programme under the SAFEXPLAIN Project (www.safexplain.eu), grant agreement num. 101069595. This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant PID2019-107255GB-C21/AEI/10.13039/501100011033. M. Caro has received funding under grant PRE2022-102440 funded by MICIU/AEI/10.13039/501100011033 and, by ESF+.

## References

[1] International Standards Organization and SAE International, *ISO/SAE PAS 22736:2021: Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles*, 2021.

[2] SAFEXPLAIN Consortium, "Space case study," https://safexplain.eu/space-case-study/, Accessed in Nov-2023.

[3] L. Lutnyk et al., "Towards pilot-aware cockpits," in *1st International Workshop on Eye-Tracking in Aviation (ETAVI)*, 2020, pp. 121–127.

[4] J. Perez-Cerrolaza et al., "Artificial intelligence for safety-critical systems in industrial and transportation domains: A survey," *ACM Comput. Surv.*, oct 2023, just Accepted. [Online]. Available: https://doi.org/10.1145/3626314

[5] M. Caro et al., "At-scale assessment of weight clustering for energy-efficient object detection accelerators," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 2022.

[6] A. Bochkovskiy et al., "YOLOv4: Optimal speed and accuracy of object detection," 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[7] Infineon, "AURIX multicore 32-bit microcontroller family to meet safety and powertrain requirements of upcoming vehicle generations," Accessed in Oct-2023, http://www.infineon.com/cms/en/about-infineon/press/press-releases/2012/INFATV201205-040.html.

[8] S. Alcaide et al., "High-integrity GPU designs for critical real-time automotive systems," in *DATE*, 2019.

[9] ——, "Software-only based diverse redundancy for ASIL-D automotive applications on embedded HPC platforms," in *DFT*, 2020.

[10] L. K. Draghetti et al., "Detecting errors in convolutional neural networks using inter frame spatio-temporal correlation," in *IOLTS*, 2019.

[11] S. Jha et al., "Exploiting temporal data diversity for detecting safety-critical faults in AV compute systems," in *DSN*, 2022.

[12] M. Caro et al., "Efficient diverse redundant DNNs for autonomous driving," in *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2023, pp. 18–27.

[13] T. Strauss et al., "Ensemble methods as a defense to adversarial perturbations against deep neural networks," *arXiv preprint arXiv:1709.03423*, 2017.

[14] L. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, 1990.

[15] Z. Gao et al., "Soft error tolerant convolutional neural networks on fpgas with ensemble learning," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 3, pp. 291–302, 2022.

[16] J. Ayerdi, A. Iriarte, P. Valle, I. Roman, M. Illarramendi, and A. Arrieta, "Metamorphic runtime monitoring of autonomous driving systems," 2023.

[17] S. Alcaide et al., "Software-only triple diverse redundancy on GPUs for autonomous driving platforms," in *50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, 2020, pp. 82–88.

[18] "Apollo, an open autonomous driving platform," http://apollo.auto/, Accessed in Oct-2023.

[19] J. Heras, "CLoDSA image augmentation library for object detection," https://github.com/joheras/CLoDSA, Accessed in Oct-2023.

[20] C. Li, "YOLOv4 tensorflow keras implementation," https://github.com/taipingeric/yolo-v4-tf.keras, Accessed in Oct-2023.

[21] T. Lin et al., "Microsoft COCO: Common objects in context," 2015.

[22] A. Bochkovskiy, "YOLOv4 on the darknet framework and publicly available weights," https://github.com/AlexeyAB/darknet, Accessed in Oct-2023.

[23] Udacity, "Udacity self-driving car driving data," Accessed in Oct-2023. [Online]. Available: https://github.com/udacity/self-driving-car

[24] F. Yu et al., "BDD100K: A diverse driving video database with scalable annotation tooling," *CoRR*, vol. abs/1805.04687, 2018. [Online]. Available: http://arxiv.org/abs/1805.04687

[25] P. Sun et al., "Scalability in perception for autonomous driving: Waymo open dataset," *CoRR*, vol. abs/1912.04838, 2019. [Online]. Available: http://arxiv.org/abs/1912.04838

[26] M. Everingham et al., "The pascal visual object classes (VOC) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, 2010.

[27] R. Padilla et al., "A survey on performance metrics for object-detection algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2020, pp. 237–242.

[28] G. Li et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[29] A. Bosio et al., "A reliability analysis of a deep neural network," in *2019 IEEE Latin American Test Symposium (LATS)*, 2019, pp. 1–6.