A Compositional Simulation Framework for Testing Adversarial Robustness of Deep Neural Networks

Youssef Maher Nader*, Mostafa Lotfy Hatab*, Mazen Mostafa Ghaleb*, Safia Medhat Bakr*,
Tasneem A. Awaad† *, Ahmed AlGanzouri†, Mohamed Abdelsalam†, and M. Watheq El-Kharashi*‡
*The Department of Computer and Systems Engineering, Faculty of Engineering, Ain Shams University, Cairo, Egypt
†Siemens EDA - Cairo, Egypt
‡Department of Electrical and Computer Engineering, Faculty of Engineering & Computer Science,
University of Victoria, Victoria Canada
Emails:{youssefmaher725, mostafaXlotfy, safiabakr12}@gmail.com, mazenghaleb@outlook.com, {tasneem.awaad.ext@siemens.com,
tasneem.awaad@eng.asu.edu.eg}, {ahmad.al-ganzouri, mohamed.abd_el_salam_ahmed}@siemens.com, watheq@engr.uvic.ca

***Abstract*—Deep neural networks (DNN) have reached impressive performance in computer vision, making them a natural choice for object detection problems in automated driving. However, DNNs used for object detection are known to be highly vulnerable to adversarial attacks. Even small changes to the input such as adding a customized noise pattern that remains invisible to the human eye, can stimulate silent prediction errors. In this study, we present a compositional simulation framework for testing the adversarial robustness of DNNs used in object detection. We demonstrate our framework with a comprehensive case study of a speed-sign detector model with two different adversarial attacks.**

***Index Terms*—Adversarial attacks, Adversarial robustness, CARLA simulator, Convolutional neural networks, Deep learning**

I. Introduction

Deep neural networks (DNNs) are key enablers for implementing functions in systems that operate in complex and unpredictable environments (self-driving cars, smart traffic systems, smart manufacturing, etc.) [1]. However, DNNs are in principle vulnerable to adversarial examples, i.e., minimal (and usually imperceptible) perturbations applied to their input that can lead to false predictions.

Formally we can define an adversarial example X_{adv} based on the benign data input X, whose label is l when passed through a model f. X_{adv} is constructed by applying some perturbation to X, as to fool the model f into labeling the data X_{adv} as l_{adv} rather than l, where $l_{adv} \neq l$. Essentially, for some distance metric $||\dots||_p$ (for example L_1 or L_2) X_{adv} can be found as in (1) [2].

$$\begin{gathered} \min_{X_{adv}} ||X - X_{adv}||_p, \ \textit{such that} \\ f(X) = l, f(X_{adv}) = l_{adv}, l \neq l_{adv} \end{gathered} \tag{1}$$

Adversarial examples can be created against all types of machine learning models (e.g., Decision Trees, Support Vector Machine (SVM)), but the real emphasis has been given to neural networks as they expose top performance in various domains including computer vision (e.g., image classification, object detection) and natural language processing.

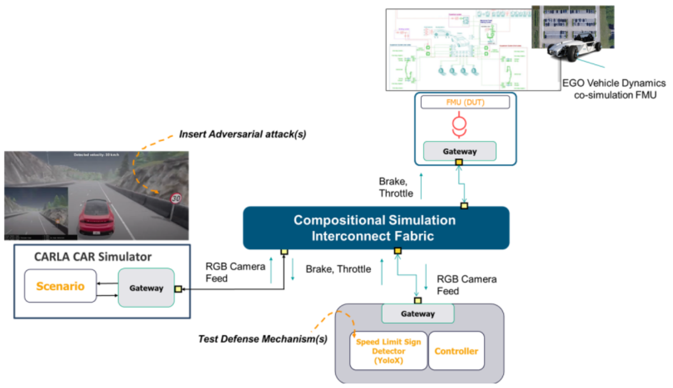

EGO Vehicle Dynamics co-simulation FMU
FMU (DUT)
Gateway
Insert Adversarial attack(s)
Brake, Throttle
Compositional Simulation Interconnect Fabric
CARLA CAR Simulator
Scenario
Gateway
RGB Camera Feed
Brake, Throttle
RGB Camera Feed
Gateway
Test Defense Mechanism(s)
Speed Limit Sign Detector (YoloX)
Controller

Fig. 1. Compositional Simulation Framework for adversarial attacks analysis

In this paper, we have used a compositional simulation interconnect fabric known as Veloce System Interconnect (VSI) [3], [4] as shown in Fig. 1. VSI functions as an intermediary layer, facilitating the linking of diverse tools and models. It enables the incorporation of models and components developed using a range of technologies, as well as the mixing of continuous and discrete modeling methods for physical components. We utilized this interconnect fabric to connect our machine learning (ML)/artificial intelligence (AI) model for perception with the Car Learning to Act (CARLA) [5] open-source scenario simulator and the complete EGO Vehicle Dynamics of the car exported as a co-simulation Functional-Mock-up Unit (FMU) [6]. The interconnect fabric provided gateways to connect these tools and models as remote clients over a backplane server.

We aim to use the assembled co-simulation framework to demonstrate the dangers of adversarial samples by implementing two adversarial attacks, which are Fast Gradient Sign Method (FGSM) [7] and Iterative Fast Gradient Sign Method (IT-FGSM) [8] on RGB camera sensor images and show how these attacks can force a You Only Look Once X (YOLOX) speed sign detector DNN model [9] to misidentify speed limit signs. In order to defend against these attacks, we implemented a defense method known as a “High-Level Guided Denoiser” (HGD) [10] in order to extract the adversarial noise from adversarial examples and send a denoised image to the detector, which allowed us to improve the detector's performance when subjected to adversarial attacks.

Our contribution is introducing a novel compositional simulation framework for online adversarial attacks analysis and defense mechanisms testing, which combines scenario simulation, vehicle dynamics, and ML/AI models (AI Agents). The framework features a visual inspection of the effect of adversarial attacks and defenses, in order to be able to tell at a glance if a given adversarial defense is successful at eliminating the dangers posed by an attack.

II. Related Work

Goodfellow et al. introduced a new family of fast methods for the generation of adversarial samples based on gradients. In their work, they introduced FGSM and used it to generate adversarial attacks, demonstrated it on several models and datasets, and finally used adversarial training to defend against this attack method [7].

Kurakin et al. introduced IT-FGSM, which is based on reapplying FGSM to the output adversarial sample multiple times. In their work, they used adversarial attacks including IT-FGSM on larger models trained on the ImageNet dataset [8].

Liao et al. introduced the concept of utilizing an HGD in order to defend against adversarial attacks. In their work, they suggested the use of a high-level representation of the data to remove adversarial noise from an adversarial sample. Furthermore, they proposed the use of a U-Network (U-Net) architecture, a convolutional neural network (CNN) whose layers are in a U-shape, as the denoiser [10].

The previous papers illustrated an offline analysis of these methods. We aim to provide an interactive real-time framework for the analysis of different attacks and defenses as we will demonstrate on the three previous methods. Furthermore, we provide a method for training an HGD model using the features extracted from a Feature Pyramid Network (FPN) rather than the method using simpler features illustrated in [10].

III. Methodology

This section details the methodology we followed in order to implement this framework and test it using different adversarial attacks and a defense method.

A. Synthetic Dataset Generation

We have used CARLA with different maps, and weather conditions to generate a dataset containing the speed limit signs with annotations (1992 images). Furthermore, a subset of The Tsinghua-Tencent 100K [11] and German Traffic Sign Detection Benchmark [12] datasets containing speed limit traffic signs were used to create a new dataset for this study, which is a dataset that contains only raw/benign images and is used to train our detector model. For the defense, another dataset, that contains malicious and benign images, was generated by applying FGSM to the previously mentioned benign datasets

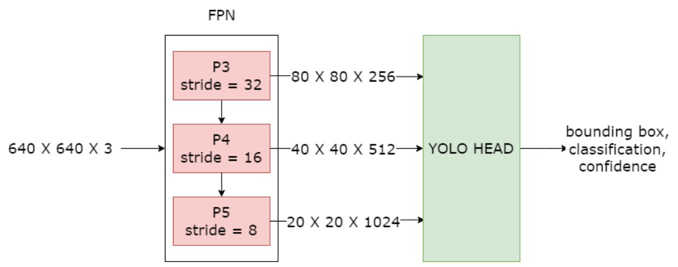


Fig. 2. Simplified architecture of YOLOX

using different values of ϵ, where ϵ is a small value defined by the user to control the amount of noise introduced in the adversarial example as shown in (2).

The training of the YOLOX detector was performed on Kaggle [16] and Microsoft Azure [15]. The PyTorch [13] framework was used for the training process. The training for the HGD was carried out on a GPU-enabled machine using an RTX graphics card with CUDA. The PyTorch framework was used for building the model and the custom training loop.

B. YOLOX Detector

To demonstrate the adversarial attacks and defenses the YOLOX-Small architecture was chosen. Subsequently, a YOLOX-Small model was trained using the previously mentioned datasets on the detection of speed limit traffic signs (the labels used correspond to $15, 20, 30, 40, 50, 60, 70, 80, 100$, and 120 speed signs). The YOLOX detector utilizes Darknet53 as its feature extractor, Darknet53 acts as an FPN giving 3 sets of feature maps labeled $P3, P4$, and $P5$ as shown in Fig. 2, these will be used in the defense mechanism as a high-level representation of each image [9].

C. Adversarial Attacks

For the purposes of this study, two attacks were used to evaluate the adversarial robustness of deep neural networks. These are FGSM, and IT-FGSM.

FGSM: This attack method aims to modify an input to obtain an adversarial example, so it maximizes the loss function when a model tries to process X. This can be represented by (2), where X is the input to the model, X_{adv} is the adversarial example, f is the model, l is the actual target label, L is the loss function, and ϵ is a small value chosen by the user.

$$X_{adv} = X - \epsilon * \frac{\partial L(f(x), l)}{\partial X} \tag{2}$$

IT-FGSM: It is a modified version of FGSM that takes the obtained adversarial example and repeats the same process with the adversarial example as an input to obtain a stronger example that gives a higher loss value.

FGSM and IT-FGSM require the actual target labels associated with the input image, which are not available at runtime. To solve this problem, a "target generator" function was implemented to obtain an approximation of labels that will be used to calculate the loss function of the model. The

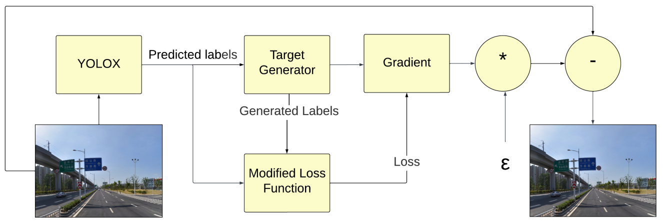


Fig. 3. Generation of an adversarial sample using FGSM at run-time. The target generator was introduced to solve the problem of lack of labels at run-time.

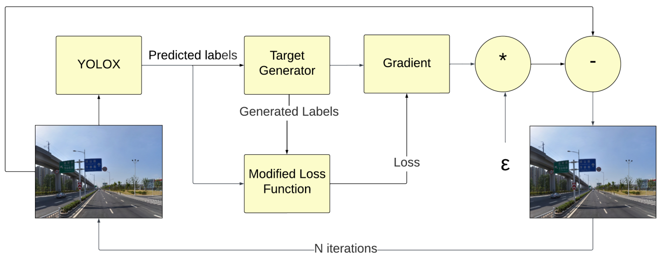


Fig. 4. Generation of an adversarial sample using IT-FGSM at run-time. The target generator was introduced to solve the problem of lack of labels at run-time.

process of obtaining the loss at run time as shown in Figs. 3 and 4 goes as follows:

- The image goes through the model to obtain the predicted labels.
- The target generator sets the generated labels based on the predicted labels, which pass a threshold of 0.5.
- The loss associated with the bounding boxes is set to 0 and the other losses are calculated using Binary Cross Entropy Loss.

D. Defense Mechanism

We chose to utilize a high-level guided denoising network [10] as the defense mechanism for this study. Specifically, a denoising U-Net architecture using dense blocks [14] was used to remove the adversarial noise added by the FGSM and IT-FGSM attacks.

To train this network, a new dataset of adversarial samples was created by applying the FGSM attack to each image in the dataset using the ϵ values$0, 1, 2, 3, 4,$ and 5. The denoising network was trained to find the adversarial noise found in an adversarial sample and output that noise. The obtained noise can be subtracted from the adversarial sample to obtain a "cleaned" version, which is fed to the speed limit traffic sign detector as shown in Fig. 5. At first, we attempted to use the U-Net architecture [10] shown in Fig. 6, however, it became obvious that training that network was infeasible with the YOLOX model and the size of the input images; due to memory and processing constraints. This prompted us to implement improvements to both the architecture and the training process. The first of these improvements was the introduction of the width parameter, which effectively reduced the number of filters in each convolutional layer, thus reducing memory consumption. Secondly, the architecture was changed

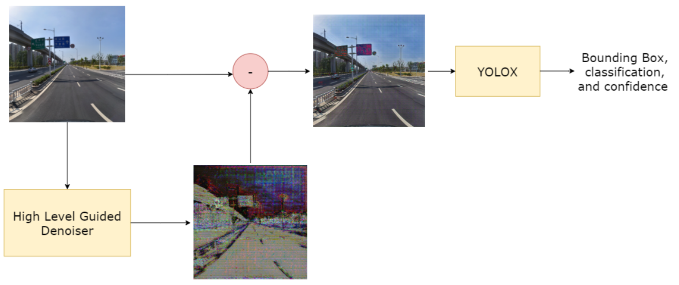

-
YOLOX
Bounding Box, classification, and confidence
High Level Guided Denoiser

Fig. 5. The denoising process

from the one shown in Fig. 6, whose building blocks can be found in Figs. 7 and 9, to a Dense block [14] based U-Net architecture shown in Fig. 8. The building blocks for this architecture can be found in Figs. 9 to 11. Furthermore, to provide clarity regarding some of the terms used in our approach, it's worth noting that "Batch Normalization (BN)" stabilizes and accelerates neural network training by normalizing each layer's outputs to have a mean of zero and a variance of one. This normalization helps maintain stable gradients during training. "Rectified Linear Unit (ReLU)" is an activation function employed in neural networks, introducing non-linearity by outputting the input if it's positive, and zero otherwise. "Convolution Layer (Conv)" is a mathematical operation in CNNs that employs a small filter or kernel to slide over the input data, producing a feature map that aids in feature detection. Finally, "Dropout2D" serves as a regularization technique designed specifically for convolutional neural networks, randomly setting a fraction of the feature maps in a layer to zero during training to prevent overfitting. By incorporating these techniques and optimizations, we aimed to make our model more efficient and capable of handling the challenges posed by the YOLOX model and large input images.

The main advantage of dense blocks is that each dense layer adds a fixed number of channels given an input with any number of channels and this reduces the number of operations performed by the model during training and inference time. The other advantage is that between each dense layer and all following dense layers within a dense block there exist skip connections, which allows for a better gradient when backpropagating and enables faster training. We added a stem to the network to aggressively downsample the image before any processing occurs to alleviate the memory problem. This downsampling is met with an opposite upsampling at the end of the network. Finally, we added gradient accumulation to reduce the memory and processing problems.

In the first architecture, the number of channels is controlled by a user-defined hyperparameter called "width", which is multiplied by the number of channels produced by each block in the network. For example, if the width = 1.0 then the number of channels produced by each block will be the same as in Fig. 6 and if width = 0.5 the number of channels will be halved. The number of channels in the second architecture is similarly controlled by the "width" parameter, in addition to the parameters "bottleneck size (bn_size)" and

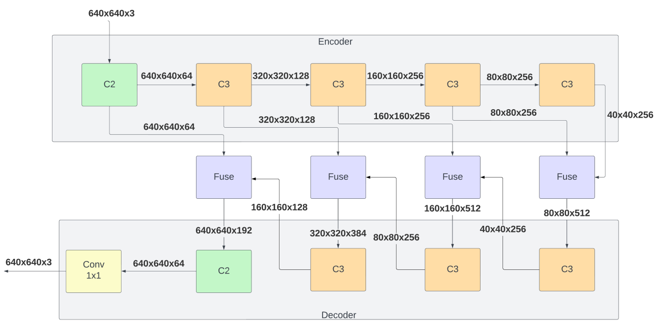


Fig. 6. U-Net denoiser architecture

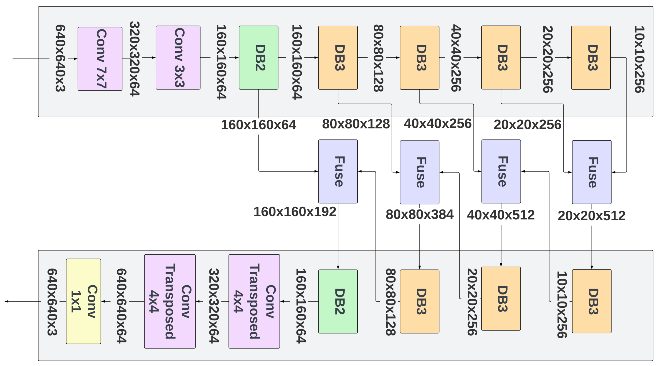


Fig. 8. Dense Block based U-Net denoiser architecture. DB2: Dense Block 2, and DB3: Dense Block 3.

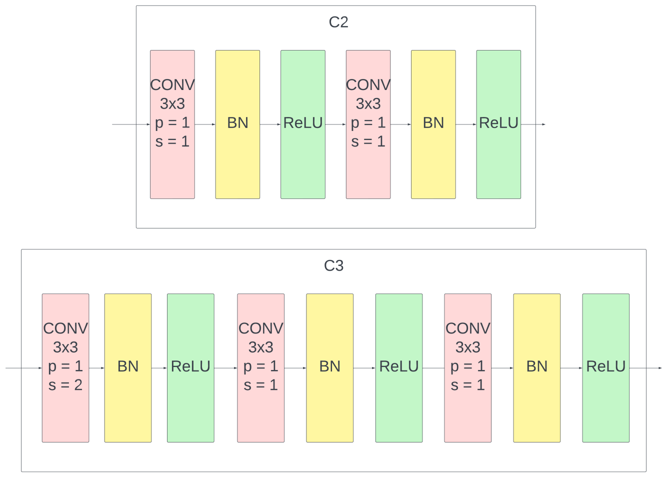
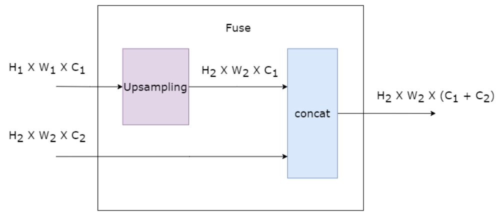


Fig. 7. U-Net denoiser building blocks



Fuse
$H_1 \times W_1 \times C_1$
Upsampling
$H_2 \times W_2 \times C_1$
concat
$H_2 \times W_2 \times (C_1 + C_2)$
$H_2 \times W_2 \times C_2$

Fig. 9. Fuse layer

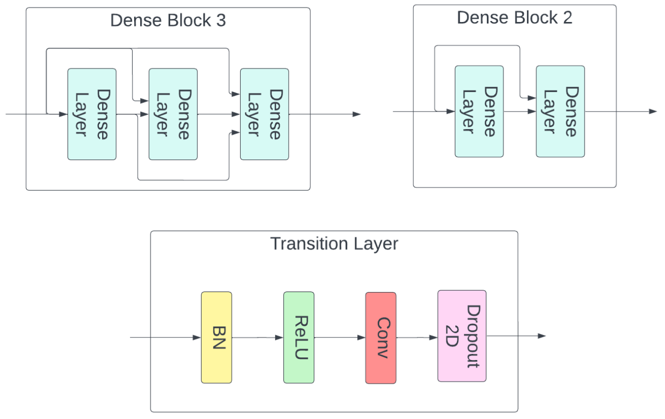


Fig. 10. Dense block based U-Net denoiser architecture building blocks. The transition layer is found between each two dense blocks to match the desired dimensions and channels of the next dense block.

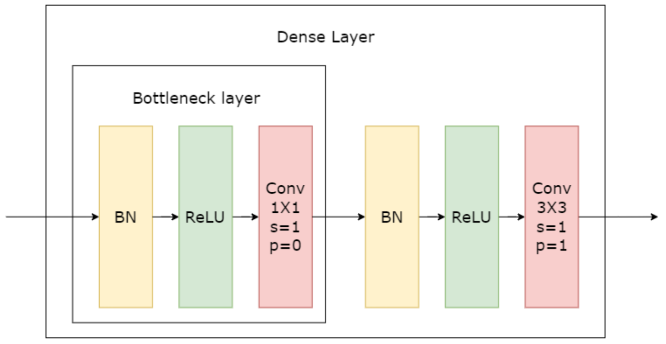


Fig. 11. Dense layer

"growth_rate". The bottleneck effect is a property of CNNs whereby by utilizing bottleneck layers the dimensions of the input feature maps are reduced, this is done in an effort to reduce the computational and memory requirements while retaining essential features from the input data. It allows the network to capture high-level abstractions by gradually reducing the spatial dimensions before expanding them in subsequent layers. The "growth_rate" determines the number of channels produced by each dense layer inside the dense block; for example, if growth_rate = 32 then each dense layer will produce an additional 32 channels from the input. This allows for a linear growth rate in the number of computations as we go deeper into the model rather than the exponential rate of growth present in the first architecture. The bn_size and growth_rate control the number of channels produced by the bottleneck layer inside the dense layer. The number of channels is equal to $growth\_rate * bn\_size$ (i.e., it controls the intensity of the bottleneck effect done by the bottleneck layer).

The training of an HGD network, as shown in Fig. 12, requires a custom loss function. The loss is dependent on the concept of finding a high-level representation of the data and penalizing the model if the image created by removing the noise (i.e., the denoised image) results in a high-level representation that is far from the high-level representation of the benign image according to some distance metric. In essence, to utilize this denoiser we need to find a representation of the data, H, and a denoising network, D, such that

$$||H(D(X_{adv}) - H(X)||_p \simeq 0. \tag{3}$$

Since we are using the YOLOX-Small architecture as a detector, we utilized the $P3$, $P4$, and $P5$ features shown in Fig. 2 from the YOLOX model as our representation of the data. We pass the benign image and denoised image to the detector model, which gives us a pair of $P3$, $P4$, and $P5$ features, which are used in the loss function as detailed in (6), (7). Additionally, we added two regularization terms to the loss. One of which is to penalize the model for outputting a denoised image, whose pixel values are greater than 255 or smaller than 0. The second regularization term is used to penalize the model for outputting noise values that are greater than 255 or smaller than -255. Both these regularization terms are used in order to ensure that the denoised image's pixel values are within the range of 0 to 255 that is normally used for pixel values. Both these terms can be found in (4), (5).

As mentioned previously, during training both the benign and denoised images are passed to the detector in order to obtain their $P3$, $P4$, and $P5$ features, the previously mentioned loss can then be calculated and the denoiser can be improved by backpropagation through the loss (i.e., by reducing the distance between the high-level representation of the denoised image and the clean image). This essentially turns the loss function into the form found in (4) to (7).

$$L_{noise} = \frac{1}{N} * \left(\sum_{i=0, y_i < -255}^{i<640*640*3} y_i + \sum_{i=0, y_i > 255}^{i<640*640*3} y_i \right)^2, \tag{4}$$

$$L_{denoised} = \frac{1}{N} * \left(\sum_{i=0, x_i < 0}^{i<640*640*3} x_i + \sum_{i=0, x_i > 255}^{i<640*640*3} x_i \right)^2, \tag{5}$$

$$\begin{aligned} L_{features} = \frac{1}{N} * (&||P3_{benign} - P3_{denoised}||_1 + \\ &||P4_{benign} - P4_{denoised}||_1 + \\ &||P5_{benign} - P5_{denoised}||_1 +), \end{aligned} \tag{6}$$

$$L_{total} = L_{features} + \lambda * (L_{denoised} + L_{noise}), \tag{7}$$

where y_i is the ith pixel in noise, x_i is the ith pixel in the denoised image, $P3$, $P4$, and $P5$ are the high-level representations of the image obtained from the feature extractor, λ is the regularization factor, $|| \ldots ||_1$ is the $L1$ norm, and N is the number of images in a training batch.

E. CARLA Environment

In order to simulate the model, attack, and defense approaches. CARLA open-source simulator for autonomous driving research was used to provide a virtual real-time environment in order to allow us to test the adversarial attack and defense methods visually.

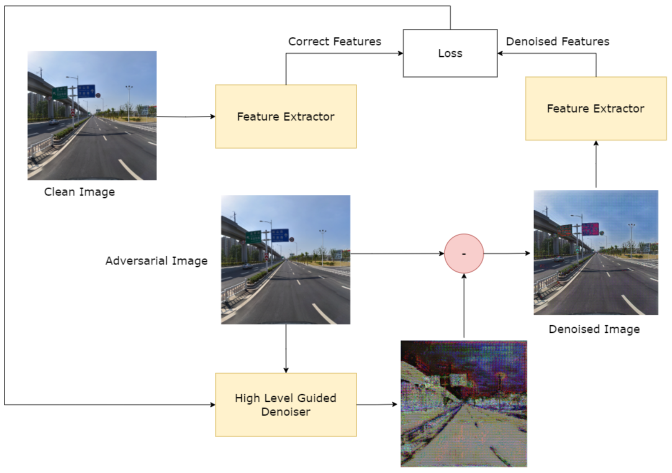


Fig. 12. The training process of the denoiser

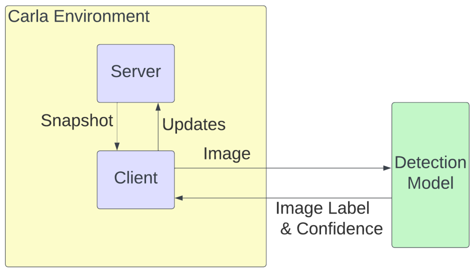


Fig. 13. The normal scenario

The CARLA environment interacts with the detection model in different scenarios, where each scenario is shown in Figs. 13 to 16. In the normal scenario, as demonstrated in Fig. 13, the CARLA environment sends an image to the detection model, which in turn returns the detections found in the sent image.

To simulate the attack scenario, as demonstrated in Fig. 14, we assume the existence of an attacker, who is capable of hijacking the image sent to the detector, where they apply some attack method and forward the perturbed image to the detection model whereby the detector will be fooled into returning incorrect detections to the CARLA environment.

In the defense scenario, as demonstrated in Fig. 15, the

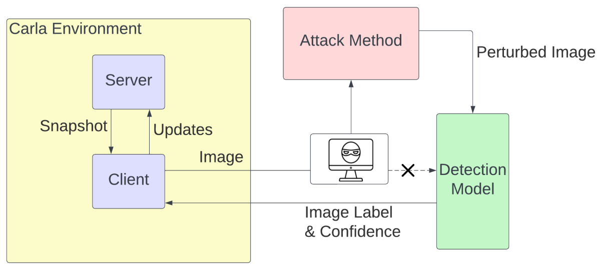


Fig. 14. The attack scenario

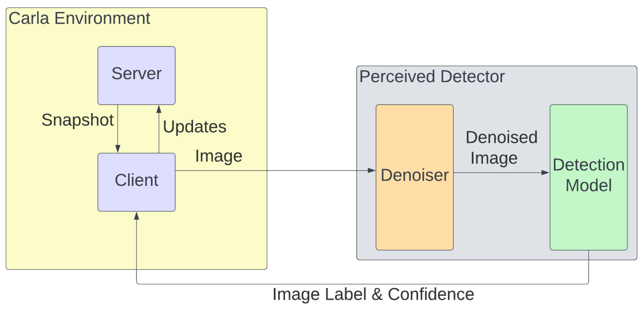


Fig. 15. The defense scenario

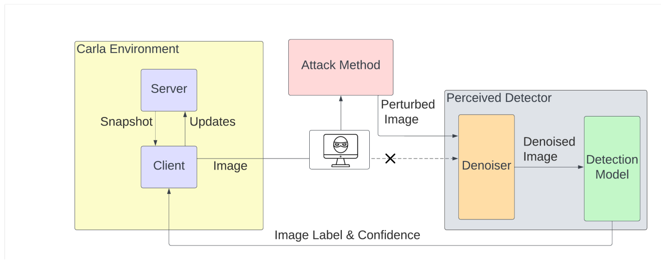


Fig. 16. The defended attack scenario

CARLA environment will send the image to what it perceives as the detector but in fact, it would be sending it to the denoiser, which would remove any present adversarial noise (in this case no adversarial noise exists), and then be passed to the detector, which in turn returns the detections to the CARLA environment.

Finally, when both the attack and defense are engaged, as demonstrated in Fig. 16, the attacker would still hijack the image, apply an attack method on it, and forward it to what they perceive to be the detector, the denoiser, in this case, will remove the adversarial noise added by the attacker allowing the detector to carry out the detection and return it to the CARLA environment.

We have implemented multiple features including the ability to run the detector, attack, and defense simultaneously, which gives us the result of the detection before applying the attack and defense, detection after the attack, and detection after the defense on the attack chosen, if any. Additionally, we have implemented another way (decoupled mode) to run the framework, where we can choose a specific scenario and get the direct output of it, for example, getting only defense detection without getting the detection after the attack or normal detection. This allows using the simultaneous mode to carry out an inspection of the results to visualize in real-time what happens under normal conditions, how the attack changes the detection, and how well the defense performs to try to remove the effect of the attack and return to the normal scenario. On the other hand, the decoupled mode allows us to simulate reality better by having only one scenario running and therefore has better computation time. We have also implemented an auto speed-limit mode, which limits the vehicle speed according to the label of the sign detected. The vehicle will follow the speed detected in the highest priority case (The priorities for the scenarios from highest to lowest are as follows, defense, attack, then detection) if it is in the simultaneous mode, otherwise, it will follow the speed detected in the scenario of the decoupled mode. The scenario is computed by receiving an image of the view above the camera with certain limits to provide a view of the street as if the camera is centered on top of the vehicle. Using that image, depending on the scenario, the required computations are made.

IV. Experimental Results

In this section, we demonstrate the effectiveness of our implemented attacks and defense. For this purpose, we used two metrics namely, precision and recall. For both metrics, we tested on the unseen portion of the dataset. We assumed that the detections generated from the YOLOX detector are the ground truth for all future calculations in order to better measure the performance of the attacks and defense without being concerned with the results from the detector (i.e., by assuming the detections from YOLOX to be the ground truth, we eliminate the performance of the detector from the calculation of the performance of the attacks and defense). Both metrics used two threshold values, the confidence threshold, and the Intersection over Union (IoU) threshold. The confidence threshold is used to reject any low-confidence detections, while the IoU threshold is used to identify if two bounding boxes are the same (i.e., when comparing a bounding box to the ground truth bounding box if their IoU score is lower than the IoU threshold, we assumed that this is not a true positive (TP) even if they have the same label).

For precision, we generated the detections using the YOLOX detector (either on attacked data, denoised benign data, or denoised adversarial data), and then we calculated of all the detections on the test data how many correspond with the bounding boxes found in the ground truth detections. Subsequently, we got a percentage of how many of the test bounding boxes have a corresponding bounding box in the ground truth detections out of all test bounding boxes, this percentage is the value of precision.

For recall, we use the same generated detections, and for each bounding box in the ground truth detections, we checked whether the generated detections have a bounding box that corresponds with the ground truth bounding box. A formal description of the metrics used can be found in (8).

$$\begin{aligned} d_f &= f(X) \\ d_s &= f(s(X)) \\ precision &= \frac{TP}{count(d_s)} \\ recall &= \frac{TP}{count(d_f)} \\ where\ TP: label(d_s) &= label(d_f), IoU(d_f, d_s) \geq 0.75, \\ & confidence(d_s) \geq 0.8, \end{aligned} \tag{8}$$

TABLE I
THE PRECISION AND RECALL RESULTS OF THE SYSTEM BEFORE AND AFTER APPLYING ATTACKS AND DEFENSE MECHANISM ON THE REAL-WORLD DATASET

Category	Precision	Recall
Denoised Benign	0.9580	0.9763
FGSM	0.3416	0.1899
Denoised FGSM	0.7392	0.7168
IT-FGSM	0.0030	0.0839
Denoised IT-FGSM	0.7067	0.6994

TABLE II
THE PRECISION AND RECALL RESULTS OF THE SYSTEM BEFORE AND AFTER APPLYING ATTACKS AND DEFENSE MECHANISM ON THE CARLA DATASET

Category	Precision	Recall
Denoised Benign	1.0000	1.0000
FGSM	0.7317	0.8604
Denoised FGSM	0.8857	0.8994
IT-FGSM	0.0253	0.4123
Denoised IT-FGSM	0.1881	0.7695

where f is the detection model, d_f is the detections made by the model on the benign data (assumed to be the ground truth), s is the scenario that is running (i.e., attack or defense or any combination of them), and d_s is detections made by the model after applying the scenario.

A. Precision and Recall Metrics

1) Results on Real-World Dataset: This section demonstrates the results of the attacks and defense of precision and recall. For this section, the detector and denoisers used were the ones trained on the real-world dataset (the test dataset consisted of 803 images).

From Table I, it can be noticed that the detector model was not robust to adversarial attacks as both the FGSM and IT-FGSM attacks are able to significantly reduce the precision and recall of the model. Additionally, it can be observed that the defense is able to remove the adversarial noise from the attacks the model was trained on (seen attacks) like FGSM. Also, it is able to remove attacks that weren't used in the training (unseen attacks) like IT-FGSM successfully (i.e., the defense is able to generalize on unseen attacks). This does not come at a significant cost for the precision and recall for denoised benign detections.

2) Results on CARLA Dataset: This section demonstrates the results of the attacks and defense of precision and recall. For this section, the detector and denoiser used are the ones trained on the CARLA simulator dataset (the test dataset consisted of 304 images). As shown in Table II, the precision and recall results of the system before and after applying attacks and defense mechanism on the CARLA dataset.

3) Results on the Mixed Dataset: This section demonstrates the results of the attacks and defense of precision and recall.

TABLE III
THE PRECISION AND RECALL RESULTS OF THE SYSTEM BEFORE AND AFTER APPLYING ATTACKS AND DEFENSE MECHANISM ON THE MIXED DATASET

Category	Precision	Recall
Denoised Benign	0.9874	0.9924
FGSM	0.5619	0.5065
Denoised FGSM	0.8785	0.8739
IT-FGSM	0.0114	0.2293
Denoised IT-FGSM	0.8624	0.8609

TABLE IV
THE HYPERPARAMETERS OF EACH OF THE DENOISER MODELS

Model	width	bn_size	growth_rate
Model 1 (Real-World Data)	1.0	4	32
Model 2 (CARLA Data)	1.0	4	32
Model 3 (Real-World and CARLA Data)	0.5	2	16

The detector and denoiser are trained on a mixed dataset, meaning both CARLA and real-world datasets (mixed test dataset consisted of 1107 images). Additionally, this denoiser is computationally faster as it uses smaller values for width, bn_size, and growth_rate. As shown in Table III, the denoiser is capable of providing satisfactory results for both seen and unseen attacks, with very little compromise in the precision and recall of denoised benign data. Moreover, the computational speed of this detector is enhanced by utilizing reduced values for width, bn_size, and growth_rate.

B. Timing for the Models

Each denoiser's timing is different depending on the values used for width, bn_size, and growth_rate. A detailed description of the values used for each denoiser can be found in Table IV. Furthermore, the timing that is found within CARLA is highly dependent on the map from which the timing was retrieved, with more graphically intensive maps requiring more processing on the GPU thus limiting the processing power that is available to the denoiser. All the presented timing results were recorded on a system fitted with Ryzen 5 3600X CPU and an NVIDIA RTX 2060 Super GPU.

As shown in Table V, there is a very minute difference between detection times on all three of the models, this is due to the same architecture being used for all three datasets. Moreover, the first two denoisers add a significant computational time overhead in order to denoise the images before they are passed to the detector. This overhead is significantly reduced in the third model as it uses a smaller value of width, bn_size, and growth_rate (i.e., the model is easier to compute). This overhead could be further reduced by further reducing the values of width, bn_size, and growth_rate, however, this may come at the cost of denoising performance.

TABLE V
THE TIMING RESULTS FOR THREE OF THE DENOISER AND DETECTOR MODELS

Model	Undefended Detection Timing	Defense and Detection Timing
Model 1 (Real-World Data)	0.0441s	0.1460s
Model 2 (CARLA Data)	0.0432s	0.1498s
Model 3 (Real-World and CARLA Data)	0.0435s	0.0937s

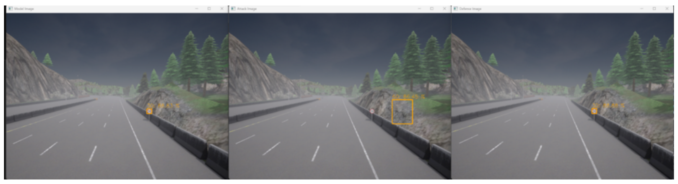


Fig. 17. Results from CARLA from left to right under normal conditions, under FGSM attack, and finally under both defense and attack

C. Visual Results Inspection

To demonstrate the behavior of the detector when it is under normal conditions, attack conditions, and when the defense is applied to remove adversarial noise introduced by the attack, we used our framework to simultaneously run all three cases in order to enable real-time comparison of the performance of the detector under these conditions and to demonstrate the results visually. In Fig. 17, we demonstrate the visual results for the FGSM attack, while Fig. 18 demonstrates the visual results for the IT-FGSM attack. It can be observed in the IT-FGSM case that the defense was able to remove the multiple erroneous bounding boxes introduced by the attack.

V. Conclusion

In this study, we have demonstrated a compositional simulation framework. That framework was introduced to perform an online verification of the enhanced defense mechanism added to guarantee the robustness of YOLOX against FGSM and IT-FGSM attacks, which required the introduction of a target generator to solve the problem of lack of labels in order to generate the attacks at runtime. Furthermore, we verified the effectiveness of our defended YOLOX model on three different datasets. In addition, we generated and annotated a synthetic dataset that can be used for the detection task of speed limit signs found within the CARLA simulator. Finally, we introduced a novel method of utilizing features from an FPN in order to train an HGD model to remove adversarial noise from adversarial samples.

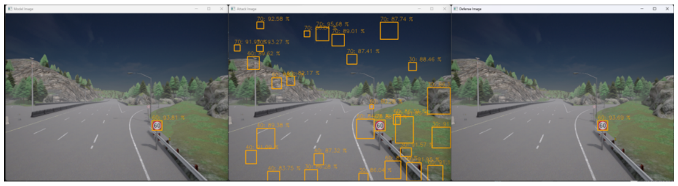


Fig. 18. Results from CARLA from left to right under normal conditions, under IT-FGSM attack, and finally under both defense and attack

References

[1] Balasubramaniam, A., & Pasricha, S. (2022, January). Object Detection in Autonomous Vehicles: Status and Open Challenges. *arXiv preprint arXiv:2201.07706*. Retrieved from https://doi.org/10.48550/arXiv.2201.07706

[2] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*. Retrieved from https://doi.org/10.48550/arXiv.1312.6199

[3] Siemens EDA. (2023). PAVE360-VSI: Veloce System Interconnect®. Retrieved from https://eda.sw.siemens.com/en-US/ic/veloce/

[4] Temperekidis, A., Kekatos, N., Katsaros, P., He, W., Bensalem, S., AbdElSabour, H., AbdElSalam, M., & Salem, A. (2022, October). Towards a Digital Twin Architecture with Formal Analysis Capabilities for Learning-Enabled Autonomous Systems. In *International Conference on Modelling and Simulation for Autonomous Systems* (pp. 163-181). Cham: Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-031-31268-7_10

[5] CARLA Open-source simulator for autonomous driving research (Version 0.9.12).(2021). Retrieved from https://carla.org/

[6] Siemens PLM Software. (2023). Simcenter Amesim. Retrieved from https://www.plm.automation.siemens.com/en/products/lms/imagine-lab/amesim/

[7] Goodfellow, I., Shlens, J., & Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015 - Conference Track)* (pp. 1–11). Retrieved from https://doi.org/10.48550/arXiv.1412.6572

[8] Kurakin, A., Goodfellow, I., & Bengio, S. (2016). Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*. Retrieved from https://doi.org/10.48550/arXiv.1611.01236

[9] Ge, Z., Liu, S., Wang, F., Li, Z., & Sun, J. (2021). Yolox: Exceeding YOLO series in 2021. *arXiv preprint arXiv:2107.08430*. Retrieved from https://doi.org/10.48550/arXiv.2107.08430

[10] Liao, F., Liang, M., Dong, Y., Pang, T., Hu, X., & Zhu, J. (2018). Defense against adversarial attacks using high-level representation guided denoiser. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1778-1787). Retrieved from https://doi.org/10.1109/cvpr.2018.00191

[11] Zhu, Z., Liang, D., Zhang, S., Huang, X., Li, B., & Hu, S. (2016). Traffic-sign detection and classification in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2110-2118). Retrieved from https://doi.org/10.1109/cvpr.2016.232

[12] Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M., & Igel, C. (2013, August). Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *The 2013 international joint conference on neural networks (IJCNN)* (pp. 1-8). Retrieved from https://doi.org/10.1109/ijcnn.2013.6706807

[13] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[14] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700-4708). Retrieved from https://doi.org/10.1109/cvpr.2017.243

[15] Microsoft. (2023). Microsoft Azure. Retrieved from https://azure.microsoft.com/

[16] Kaggle. (2023). Kaggle. Retrieved from https://www.kaggle.com/