ADVERSARIAL ATTACKS AND DEFENCES ON AUTONOMOUS VEHICLES

Ignas Vaicius
School of Computing
Newcastle University
Newcastle upon Tyne, NE1 7RU
i.vaicius1@newcastle.ac.uk

May 16, 2023

ABSTRACT

The growing reliance on Deep Neural Networks (DNNs) in critical applications, especially autonomous vehicles, has led to the necessity of addressing adversarial attacks. These attacks can cause DNN-based systems to make incorrect decisions, resulting in serious consequences in real-world scenarios. This dissertation investigates and develops effective adversarial attack and defence techniques for real-time object detection models in autonomous vehicles, while also illustrating the processing of targeted inputs by the adversarial attack and defence methods A custom YOLOv5n model was trained on the KITTI dataset, and its parameters were further evolved. The implemented attacks included the Fast Gradient Method (FGM), Projected Gradient Descent (PGD), and Adversarial Patch (AP). FGM and PGD proved effective, while AP did not yield positive results. However, the images created by these attacks were noticeably different. For defences, Adversarial Training (AT), Feature Squeezing (FS), and Spatial Smoothing (SS) were employed. FS was deemed ineffective and SS showed limited results as standalone defensive methods, while AT significantly increased the model's robustness in various scenarios. The study concludes with an evaluation of the proposed attack and defence methods and identifies areas for future research and potential improvements.

1 Introduction

In today's world, Deep Neural Networks (DNNs) play a vital role in various aspects of our lives, ranging from critical healthcare systems and financial models to autonomous vehicles. The increasing prevalence of autonomous vehicles in recent years has led to a corresponding rise in adversarial attacks targeting their underlying DNNs. Ensuring the reliability and safety of DNN-based systems is crucial, making the development of robust defences against such attacks a priority. Detecting and mitigating adversarial attacks in real time is challenging, as it requires high computational efficiency and low latency. Thus, research on real-time defences and attacks on DNNs is essential for advancing new and strengthened techniques to identify and protect against adversarial attacks. Given the risks associated with adversarial attacks on autonomous vehicles, it is vital not to overlook their potential consequences. Despite the complexity of models used in autonomous vehicles, they are not immune to errors that can lead to severe or even fatal accidents, as evidenced by incidents such as the 2018 fatal accident involving a pedestrian [14]. Furthermore, research has demonstrated that adversarial attacks can cause vehicles to behave erratically [6]. Although some techniques have been proposed to make models more robust, the challenge remains to find effective defence methods that perform well in high-computational efficiency and low-latency environments. This research paper aims to investigate and develop visible adversarial attack methods and implement effective defence mechanisms to address these challenges. Specifically, the objectives of this project are:

- **Develop three distinct adversarial attack methods - FGM, PGD, and AP** - with visible transformations to demonstrate what fools the model when the attacks are invisible.
- **Implement three defence methods - AT, FS, and SS** - to test and showcase their standalone effectiveness when dealing with the attacks.

- **Evaluate the performance of the implemented attack and defence methods in a real-time environment**, focusing on their computational efficiency and accuracy.

- **Complete the research**, development, and evaluation of the proposed methods within the time frame of the dissertation, ensuring timely submission of the final document.

The structure of this dissertation is as follows: The paper will first provide background information on the general topics related to the methods and object detection models. Then, it will detail the specific methodology and implementation of the object detection model, and attack and defence methods. Finally, it will present the results obtained from the research. By achieving these objectives, this research will contribute to the ongoing efforts to enhance the security and resilience of DNN-based systems, particularly in the context of autonomous vehicles.

2 Background review

This section will provide a comprehensive background on adversarial attacks and defences in the context of object detection models, such as YOLOv5, delving into the methodologies, goals, and challenges associated with these concepts. I will begin by exploring the foundations of adversarial attacks, detailing the underlying vulnerabilities they exploit in DNNs. Next, I will discuss adversarial defences and their objective of enhancing the robustness of DNNs, including YOLOv5, against adversarial attacks. Finally, I will address the challenges and limitations inherent in the development of robust DNN systems, emphasizing the importance of continued research in this area. This background information will set the stage for my in-depth examination of specific adversarial attacks and defences in the subsequent sections of this dissertation.

2.1 YOLOv5

2.1.1 "You Only Look Once"

YOLO (*You Only Look Once*) is a widely recognized real-time object detection algorithm, known for its small size and fast computation speed. In addition to detecting object labels using a single neural network, it can also perform classification and object prediction of bounding boxes. The algorithm was developed by Joseph Redmon et al. in 2016, who also published the research paper *"You Only Look Once: Unified, Real-Time Object Detection"* [23] that same year. In 2016, YOLOv2 [24] was proposed by Joseph Redmon, introducing new techniques to enhance the precision of prediction bounding boxes. The final version developed by Redmon, YOLOv3 [25], further improved precision and accuracy by adding more layers to the architecture and enlarging the output grid. Subsequently, YOLOv4 [33] was created by A. Bochkovskiy et al., outperforming previous YOLO models in terms of speed, precision, and accuracy through changes to its architecture and learning methodology. The YOLO algorithm was later integrated into PyTorch by researcher Glenn Jocher and his Ultralytics LLC research department, leading to the release of YOLOv5, YOLOv6, YOLOv7, and the most recent version, YOLOv8.

2.1.2 Architecture

The YOLOv5 model architecture consists of three primary components: the backbone, the neck, and the head, as illustrated in *Figure 1*.

[H]

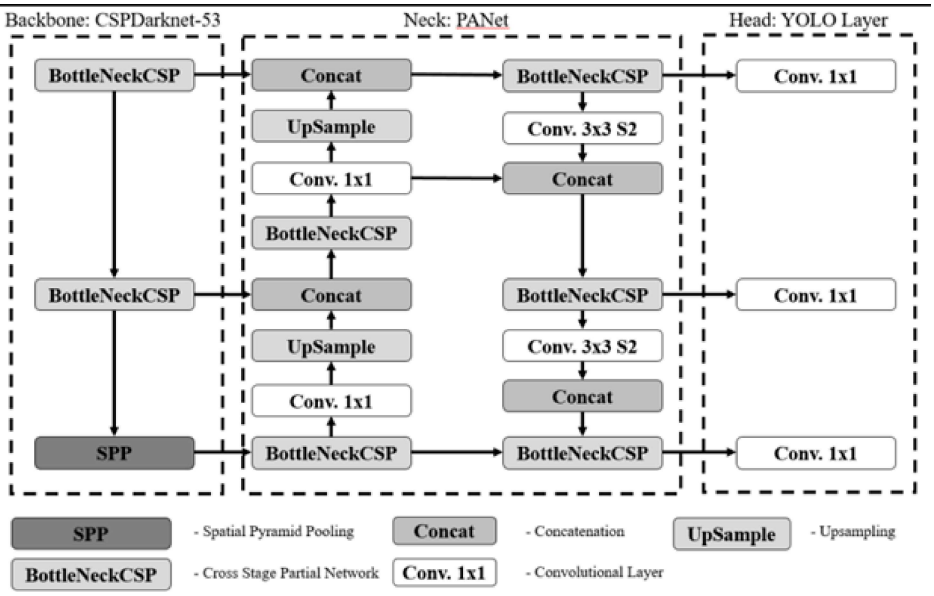


Figure 1: YOLOv5 architecture [28].

The backbone of YOLOv5 is responsible for extracting features from the input images that are crucial for object identification. This process is carried out using *CSPDarknet53*, a Darknet neural network that employs cross-stage partial connections. The *CSPDarknet53* consists of 53 convolution layers and incorporates *BottleNeckCSP* as residual blocks, which not only determine the specific model of YOLOv5 (e.g., YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x) but also help the model tackle the vanishing gradient problem. The *Spatial Pyramid Pooling (SPP)* module is used to increase the region of the input image that a given neuron in the network is sensitive to and capture features of different scales [36]. The neck connects the backbone and the head, transferring and processing the extracted features. In YOLOv5, the *Path Aggregation Network (PANet)* [28] is employed as the neck. PANet merges high-level (top-down) abstract features and low-level (bottom-up) meaningful features by utilizing both pathways and adaptive pooling. This results in more efficient feature fusion, which ultimately leads to improved performance in object detection tasks. The final component, the head, is responsible for generating bounding boxes and their corresponding labels. To enhance the accuracy of predictions, the head employs the *Generalized Intersection over Union (GIoU)* loss function, shown as $GIoU = IoU - (C - U) / C$, IoU – Intersection over Union, C – Minimum Enclosing Area and U being Union area. Additionally, to minimize the occlusion of bounding boxes, the head implements *Weighted Non-Maximum Suppression (WNMS)*.

2.1.3 Performance

The YOLOv5 object detection model, as illustrated in *Figure 2*, is recognized as one of the most efficient models in terms of speed. The figure presents a comparative analysis of YOLOv5 through YOLOv8 models, where YOLOv5 demonstrates the lowest *mean Average Precision (mAP)* score, yet it is the most compact among the models discussed, with YOLOv5n being the smallest in size. Furthermore, although YOLOv5 does not exhibit the highest precision, it displays comparable latency to other YOLO iterations during image inference.

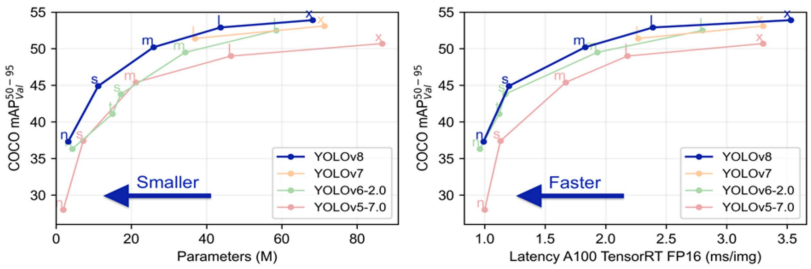


Figure 2: YOLO model comparison [13].

Metrics	YOLOv5l (Yl)	YOLOv5m (Ym)	YOLOv5s (Ys)	ResNet50 (FPN)	VGG16	MVGG16	MobileNetV2	Inception V3
Precision (P)	86.43%	86.96%	76.73%	91.9%	69.8%	81.4%	63.1%	72.3%
Training Loss	0.015	0.017	0.020	0.065	0.226	0.136	0.209	0.194
Mean Avg. Precision (mAP@0.5–0.95)	63.43%	61.54%	58.9%	64.12%	35.3%	45.4%	30.5%	32.3%
Inference Speed (Image Res. 1774x2365)	0.014 s	0.012 s	0.009 s	0.098 s	0.114 s	0.047 s	0.036 s	0.052 s
Inference Speed (Image Res. 204x170)	0.018 s	0.013 s	0.009 s	0.065 s	0.119 s	0.052 s	0.032 s	0.056 s
Training Time/Epoch	26 s	16 s	12 s	124 s	173 s	105 s	80 s	95 s
Total Training Time	31,200 s	19,200 s	14,400 s	12,400 s	17,300 s	10,500 s	8,000 s	9,500 s
Model Size (MB)	95.3	43.3	14.8	165.7	175.5	134.5	329.8	417.2

Figure 3: Comparison of YOLOv5 and other object detection models [2].

Figure 3, on the other hand, juxtaposes the performance of YOLOv5 with other prevalent object detection models. According to the data in *Figure 3*, YOLOv5 exhibits the second-highest precision (*86.96%*) coupled with the lowest training loss (*0.017-0.020*) among the models evaluated. While it does not surpass the highest *mAP* value of *ResNet50* (*64.12%*), it remains competitive with a close *mAP* value of *63.43%*. Other YOLOv5 variants range between *61.54%* and *58.9%* in *mAP*. In summary, the YOLOv5 model demonstrates superior speed and compactness as an object detection model, albeit at a slight cost to precision.

2.2 Adversarial Attacks

The advent of adversarial attacks aligns with the emergence and popularity of object detection models at the turn of the decade. These attacks can be bifurcated into two broad categories: digital adversarial attacks, designed to exploit the structure of a model, and static adversarial attacks, which generate real-world physical objects intended to deceive the system [12]. Research indicates that minor perturbations to the input image can yield misleading outputs, causing the model to produce erroneous results. Digital adversarial attacks can manifest as *white-box* or *black-box* attacks. In a white-box attack scenario, the adversarial agent has comprehensive knowledge of the target model, including its structure, training data, neural network configuration, parameters and hyperparameters, and the model's gradient loss function and prediction outcomes. Conversely, a black-box attack is more complex, as the attacker only has access to the inputs and outputs from the target model. However, the complexity of a black-box attack can be mitigated by constructing a surrogate model of the targeted DNN, subsequently enabling a white-box attack. Static image attacks resemble white-box attacks, but the generated image possesses inherent limitations in deceiving the model, as the model may adapt to the image following the initial exposure. Yet, it should be noted that all adversarial attacks adhere to the function illustrated in *Equation 1* [5], where f represents the target model, and x denotes the original image input $x \in X$ possessing a class c. The addition of a perturbation ϵ to the input image yields an adversarial image x', characterized by the perturbed image class c':

$$f(x') = x + \epsilon \tag{1}$$

In summary, adversarial attacks present an effective mechanism for generating adversarial images. The efficacy of these attacks correlates with the extent of prior knowledge about the target model's structure and the magnitude of the applied perturbation.

2.3 Adversarial Defences

In light of the increasing prevalence of adversarial attacks, a necessity has emerged for more robust DNN models. This requirement has been the catalyst for the development of adversarial defence techniques. These techniques can be broadly categorised as mitigative, detective, or eliminative, in relation to adversarial attacks. Mitigative techniques primarily involve preprocessing the input or the output of the model. One such approach is the use of data augmentation techniques [16]. This process involves the transformation of the input through various affine transformations. These transformations manipulate the image while maintaining its fundamental characteristics and key values. Techniques might include parameter reduction, image scaling or squeezing, cropping, and adjusting the image axis. Through these techniques, the model gains improved resilience when dealing with perturbed images. Detective techniques for adversarial examples involve a more direct approach. This technique trains a model on pre-existing adversarial images. As a result, during inference, the model is better equipped to identify images with malicious alterations. Eliminative techniques, which have similarities to AT methods, involve the removal of attacking images. This can be implemented by purging the images from the input before they are processed by the model, or by excluding outputs that appear to have been tampered with. However, this approach has its limitations. The removal of images can undermine the model's detection capabilities and may prompt the model to either underfit or overfit existing data. In conclusion, while effective defences against adversarial attacks do exist—whether they prepare the input image to be less susceptible to attacks, or train the model to better handle, detect, or remove altered images—no defence technique can guarantee

complete protection. This is largely due to the continuously evolving nature of adversarial attacks and the potential for attackers to customise their attacks based on the model's architecture.

3 Methodology

In this section, I will discuss the methods used for creating adversarial attacks and defences, as well as the rationale behind choosing the *Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI)* [7] dataset for these methods. I will also explain my decision to use the YOLOv5n model and the approach I employed for training it. Furthermore, I will describe the evaluation methods and metrics used in my research. This section will provide a comprehensive summary of all the methods featured in this study.

3.1 Dataset

3.1.1 KITTI dataset

The KITTI dataset, popular in the field of autonomous driving, is often used for benchmarking and training object detection models. Comprised of *7,481* training and *7,518* testing images, it totals *80,256* labelled objects intended for object detection and model training. However, labels for testing images are not publicly accessible. The KITTI dataset extends beyond standard 2D images, incorporating a multitude of parameters beneficial to object detection. These include grayscale and colour channels, visual odometry data, flow and stereo scene flow (enhancing the understanding of object motion), 3D bounding boxes, depth maps, GPS and IMU (*Inertial Measurement Unit*) data (providing insights into vehicle orientation and motion), and LIDAR point cloud data. The dataset comprises several types of objects, including *"Car", "Van", "Truck", "Pedestrian", "Person Sitting", "Cyclist", "Tram", and "Miscellaneous"*. The "Miscellaneous" category is used to label objects that do not fall into any of the standard classes. The details of the data utilized for training images are presented in *Table 1*.

Index	Field	Description
0	Type	Category of the object (e.g., Pedestrian)
1	Truncated	Indicates if the object is truncated or cut off in the image
2	Occluded	Indicates if the object is fully visible or occluded by any other object
3	Alpha	Observation angle of the object
4-7	Bounding Box	2D bounding box coordinates for the object (top left and bottom right corners)
8-10	3D Dimensions	Dimensions of the object in 3D space (height, width, and length)
11-13	3D Location	3D location of the object in the camera's coordinate system
14	Rotation_y	Rotation of the object around the y-axis in the camera's coordinate system

Table 1: KITTI Label Format with Index

The data is sourced from various specialized equipment. Grayscale and colour images are captured using two high-resolution video cameras, while ground truths are determined using a Velodyne laser scanner and a GPS localization system [8]. Images are collected in diverse, real-world environments, including urban areas and highways in the mid-sized city of Karlsruhe, Germany, to ensure a wide range of scenarios are represented. In summary, the KITTI dataset is a robust collection of various urban environment images for training and testing, complemented by additional information that can significantly enhance the performance of object detection models and related algorithms.

3.1.2 Video

In order to evaluate the model's performance in real-world scenarios, and to examine the impact of adversarial attacks and defences on the model, I opted to utilize a specific YouTube video [15]. This video features the viewpoint of a vehicle navigating through the streets of London, providing a representative and challenging environment for the model. This approach offers a practical context for assessing the robustness of the adversarial attacks and defences implemented on the custom-trained model.

3.2 YOLOv5n

In this study, the YOLOv5n model was chosen for object detection in real-time video, given the constraints of limited computational hardware. Despite having lower accuracy compared to other YOLOv5 models, as depicted in *Figure 2*, YOLOv5n stands out in terms of speed – a vital attribute for object recognition in autonomous vehicles. Moreover, the model's relative lower accuracy presents an opportunity for substantial improvement through custom training methods,

making the impacts of adversarial attacks more pronounced and noticeable during evaluations. The YOLOv5n model also benefits from comprehensive documentation on GitHub and a PyTorch-based implementation, facilitating seamless integration into the project. In essence, the selection of the YOLOv5n model was driven by its speed, computational efficiency, well-documented resources, and susceptibility to adversarial attacks, enabling a more effective visualization and evaluation of these attacks and defences.

3.2.1 Training

The training process used in the YOLOv5n model is shown in the pseudo-code in *Algorithm 1*.

Algorithm 1 Training the YOLOv5 model

```
Import necessary libraries (PyTorch, YOLOv5 etc.)
Load the preprocessed KITTI dataset
Split the dataset into training, testing and validation sets
Initialize the YOLOv5n model with pre-defined configuration
Define the loss function
Define the optimizer
for each epoch in number of epochs do
    Reset the total loss accumulator to 0
    for each batch in training data do
        Forward propagate the batch through the model
        Compute the loss
        Backpropagate the loss
        Update the model parameters using the optimizer
        Add the batch loss to the total loss
    end for
    Compute and print the average loss for this epoch
    Evaluate the model on the validation set
    Print the performance on the validation set
end for
Save the trained model
```

The first step involves importing necessary libraries such as PyTorch and the YOLOv5 model. The training process requires significant computational resources, which is why it is beneficial to use a GPU. GPU acceleration is critical for deep learning models like YOLOv5n, as it can greatly speed up computations, especially matrix multiplications that are abundant in these models.

The preprocessed KITTI dataset is then loaded and divided into training and validation sets. The YOLOv5n model is initialized with a pre-defined configuration. The loss function is defined next - YOLOv5 uses a combination of *Mean Square Error (MSE)* and *Cross-Entropy loss*. MSE is used for bounding box regression, calculating the loss between predicted and ground truth bounding box coordinates. Cross-Entropy loss, on the other hand, is used for classification - determining the class of the detected object, and objectness - determining whether there is an object or not. Cross-Entropy loss measures the performance of a classification model whose output is a probability value between 0 and 1 [19]. It is also called *Softmax Loss* if it is used for multi-labelled images.

The optimizer is then defined. This study uses the *Stochastic Gradient Descent (SGD)* optimizer because it generally provides better noise resistance, making the model more robust against adversarial attacks, as suggested by Zhou et al. (2021) [37]. Following the setup, the training process begins. For each epoch, the total loss for that epoch is initialized to 0. Then, for each batch in the training data, the batch is propagated forward through the model, and the loss is computed and then backpropagated. The model parameters are updated using the optimizer, and the batch loss is added to the total loss for the epoch. The average loss for the epoch is then computed and printed. Next, the model is evaluated on the validation set. Monitoring the model's performance on the validation set is a critical step in preventing overfitting. Overfitting occurs when a model learns to perform very well on the training data but poorly on unseen data. By evaluating the model's performance on the validation set at the end of each epoch, it is possible to keep track of how well the model is generalizing to unseen data. Finally, the trained model is saved for future use.

3.2.2 Evaluation

In assessing the performance of the YOLOv5 model, a comprehensive suite of evaluation metrics has been selected to ensure a multifaceted understanding of the model's efficacy. These metrics include the *F1 score*, *Precision* (specifically *mAP*), *Recall*, *Precision-Recall* scores, and the *Confusion Matrix*. The *F1 score*, in particular, is a robust measure of overall class classification performance. Given the unbalanced nature of the KITTI dataset, as illustrated in *Figure 4*, the *F1 score* is particularly apt for this evaluation as it can provide a more accurate representation of the model's performance.

The next metric incorporated in this evaluation is *Precision*, specifically *mean Average Precision (mAP)*. This metric is pivotal in shedding light on the model's ability to detect objects within an image accurately. Additionally, the *Recall metric* will elucidate the proportion of true positives to the total number of relevant objects, thereby providing insights into the model's sensitivity. Complementing these metrics, the *Precision-Recall score* has also been included in this evaluation. This score will highlight the trade-off between *precision* and *recall*, which is a key aspect of understanding the model's performance. Lastly, a *Confusion Matrix* will be employed to facilitate a deeper comprehension of the model's detection of true positives, true negatives, false positives, and false negatives. Collectively, these carefully selected evaluation metrics will provide a comprehensive overview of the model's performance during the training phase. Furthermore, they will allow for an insightful evaluation of the efficacy of any attacks and defences implemented in subsequent stages of this research.

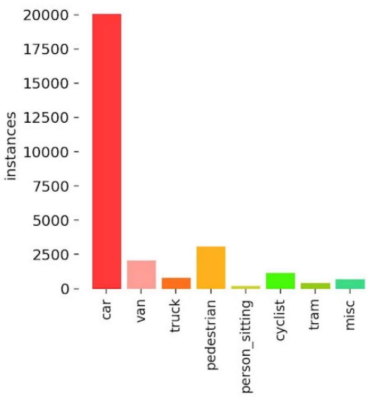


Figure 4: Label instances in KITTI dataset

3.2.3 Hardware

The importance of suitable hardware and software choices for the YOLOv5 model, as well as the subsequent attacks and defences, cannot be understated. As demonstrated in [3], utilizing a GPU for training can significantly reduce the time taken compared to a CPU-based approach, making it the preferred choice for this research. The specific hardware and software selections, along with their justifications, will be discussed in detail in the implementation section of the dissertation. This will provide a comprehensive understanding of the technologies used, while also allowing for reproducibility of the work.

3.3 Adversarial Attacks

The selection of adversarial attacks to be deployed against the object detection model was a strategic process, predicated on the desire to present a diverse array of challenges to the YOLOv5 model. Three distinct attacks, each characterized by increasing levels of intricacy in their implementation and methodology, have been chosen for this research. The first of these is the FGM, a widely recognized and effective attack that employs the Fast Gradient Sign-based method. This attack, though relatively straightforward in its application, can reveal critical vulnerabilities in the model. Secondly, the research will delve into the realm of optimization-based method attacks through the application of the PGD attack. This attack involves a more sophisticated and iterative approach, providing a more nuanced exploration of the model's resilience to adversarial manipulation. Lastly, the AP attack will be considered. This attack presents a unique challenge as it involves the generation of a maliciously crafted patch designed to provoke misclassifications. It represents one of the more complex adversarial attacks and is expected to offer a rigorous test of the YOLOv5 model's robustness. Collectively, these attacks, diverse in their complexity and methodology, are anticipated to illuminate the inherent weaknesses of the YOLOv5 model. Furthermore, they are expected to provide a robust testing ground for the efficacy of the defensive measures that will be explored in subsequent sections of this dissertation.

3.3.1 Fast Gradient Method (FGM)

FGM is a recognized adversarial attack, known for its effectiveness when applied to DNNs, including models such as YOLOv5. The decision to employ FGM in this research is based on substantial evidence from prior research [10]. These studies suggest that despite their complex structure and inherent non-linearity, DNNs can exhibit linear behaviour

between their convolutional layers and activation functions, such as ReLU. This characteristic linear behaviour becomes the fulcrum of adversarial attacks like FGM.

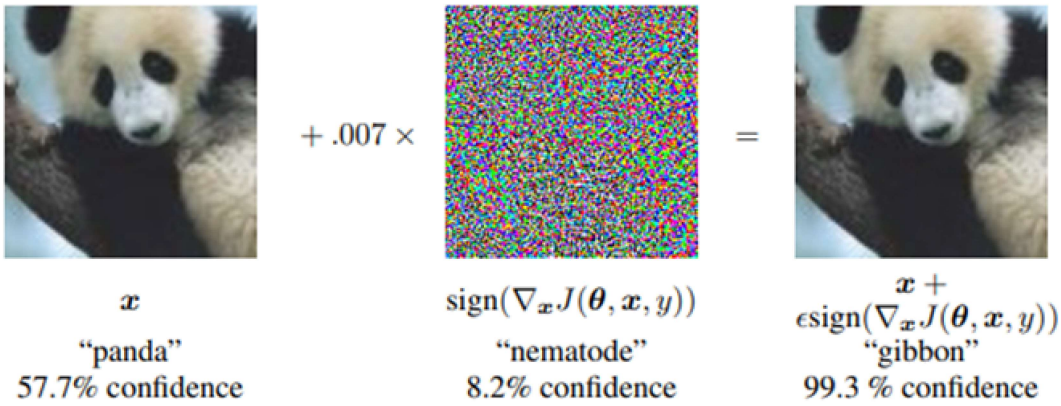


Figure 5: example of FGM attack.

$$\eta = \epsilon \, \text{sign}(\nabla_x J(\theta, x, y)) \tag{2}$$

As illustrated in *Figure 5* from [10], FGM operates by adding a perturbation, computed with respect to the loss function of the model, to the input image. This action can cause the model to misclassify the image with a high level of confidence. The perturbation η, as defined in *Equation 2*, is a function of the model parameters (θ), the input (x), the targets (y), the cost function used to train the network ($J(\theta, x, y)$), and a parameter epsilon that determines the perturbation's magnitude. The ϵ parameter is pivotal for the success of the FGM. Generally, this value must be carefully chosen to strike a balance — it should be large enough to significantly alter the model's prediction, yet small enough to remain imperceptible to the human eye. However, the aim of this research involves a slight departure from this typical objective. Here, the intention is to clearly visualize the impact of adversarial attacks, making them discernible to the naked eye. Therefore, the ideal balance is between misleading the model effectively and ensuring the image perturbations do not render the content unrecognizable. The complex process of determining the optimal ϵ value, which aligns with this research objective, will be elaborately addressed in the *Implementation* section of this dissertation. However, despite its strengths, FGM is not devoid of limitations. Primarily, it is designed to exploit the linear characteristics of models, and thus, its effectiveness could potentially diminish when applied to highly non-linear models. Furthermore, the simplicity of FGM, which is often seen as an advantage in terms of ease of implementation and lower time complexity, can be a double-edged sword. This simplicity can limit FGM's effectiveness against robust, real-time object detection models like YOLOv5, which have been trained on a diverse range of real-world scenarios and are inherently designed to handle the noise present in real-world data. In conclusion, despite these limitations, FGM serves as a powerful entry point into the realm of adversarial attacks. Its relatively straightforward implementation coupled with its capacity to expose model vulnerabilities makes it a highly valuable tool for initial investigations into the susceptibility of DNNs to adversarial perturbations. This study aims to leverage FGM's capabilities to test and analyze the robustness of autonomous vehicle technologies against adversarial attacks.

3.3.2 Projected Gradient Descent (PGD)

The next adversarial attack employed in this study is PGD, an optimization-based variant of the FGM attack. The rationale for selecting PGD is twofold. Firstly, the differential methodological approach of PGD, coupled with its ability to enhance the efficacy of FGM, enables a more meaningful comparison of the two attack strategies. Secondly, as suggested by [22], DNNs trained on PGD adversarial examples demonstrate heightened robustness against a wider variety of attacks. PGD is designed to tackle a complex optimization task, known as the '*saddle point problem*'. The objective of this problem is to ascertain the optimal set of model parameters (θ) that simultaneously enhance DNN robustness and susceptibility to adversarial attacks, as expressed in *Equation 3*.

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y)\sim D}\left[\max_{\delta \in S} L(\theta, x+\delta, y)\right]. \tag{3}$$

In *Equation 3*, θ represents the model parameters subject to optimization, while $\rho(\theta)$ denotes the computed loss of the model, which is aggregated from all distributions D of available input and output pairs (x, y). The function $L(\theta, x+\delta, y)$ signifies the loss function applied to a perturbed image, with $x+\delta$ representing the adversarial example incorporating a small perturbation δ.

By inverting the loss function, PGD transforms a minimization problem into a maximization problem, thereby operating as a '*negative loss function*'. This concept is illustrated in *Equation 4*.

$$x^{t+1} = \Pi^{x+S}(x^t + \alpha \operatorname{sgn}(\nabla_x L(\theta, x, y))). \tag{4}$$

Equation 4 presents the iterative process of PGD, with x^{t+1} being the new iteration of the adversarial image. The perturbation set is represented by S, and Π^{x+S} ensures that subsequent perturbations remain within the defined set. The variable x^t symbolizes the current step, while α denotes the step size, which determines the perturbation magnitude in relation to the gradient. The final segment, $\operatorname{sgn}(\nabla_x L(\theta, x, y))$, identical to the one used in the FGM method, dictates the direction of the perturbation in relation to the input and its loss. The iterative approach of PGD generates adversarial examples that are more effective than those produced by FGM.

However, PGD is not without limitations. Firstly, it is computationally more demanding than FGM, with the processing time increasing proportionally to the selected iterations. Secondly, PGD necessitates detailed knowledge of the model's architecture, particularly the loss function. While this does not pose a challenge for this project due to its white-box attack orientation, complications may arise with unknown models. Lastly, PGD, like other optimization-based attacks, is vulnerable to the local minima problem, as discussed in [4]. This issue arises when the algorithm, in its quest to maximize loss, identifies a perturbation that locally maximizes the loss, thereby terminating the process even though a different perturbation within the ϵ-defined space could potentially result in a higher loss.

Despite these limitations, PGD was chosen for its ability to improve upon FGM, its demonstrated success in deceiving a multitude of DNN-based object detectors, and its utilization of an optimization function that bolsters defences against a wide array of adversarial attacks.

3.3.3 Adversarial Patch (AP)

The final adversarial approach discussed in this dissertation is the AP attack. This attack generates universal patches that can lead a DNN model to misclassify or mislabel objects in the input image. The rationale for its inclusion lies in its recent application on a YOLOv3 model, as reported in [18]. This research aims to investigate whether the AP attack retains its effectiveness when deployed against the more advanced YOLOv5 model. The AP attack diverges from the aforementioned methods by applying a single patch to the image, rather than applying a gradient across the entire image, in an attempt to deceive the object detection model. *Equation 5* provides a representation of how a patch is implemented.

$$\arg\max_{\delta} E_{(x,y)\sim D, t\sim T}[J(h_\theta(A(\delta, x, t)), y)] \tag{5}$$

In the function being discussed, the term $\arg\max_\delta$ indicates that the function will yield the maximum δ values. These maxima are the adversarial patches. The expression $E_{(x,y)\sim D,t\sim T}$ signifies the expectation over x (the input image) and y (the corresponding label), where D represents a distribution over the data samples, and T denotes a distribution over possible patch transformations.

J signifies the loss function, h_θ represents the DNN model parameterized by θ, and A symbolizes the "*patch application function*". This function, when applied to (δ, x, t), implement the AP to the input, where δ is the patch, t is the transformation, and x is the image. The patch is applied by masking the appropriate pixels in the image.

$$\delta := \operatorname{clip}_{[0,1]}(\delta + \alpha \cdot \operatorname{sign}(\nabla_\delta J(h_\theta(A(\delta, x, t)), y))) \tag{6}$$

The ϵ value, which is represented by δ, is computed as per *Equation 6*. This function constructs the AP δ by employing the PGD method for generating the next iteration of the adversarial image $\operatorname{sign}(\nabla_\delta J(h_\theta(A(\delta, x, t)), y))$. However, it deviates from the conventional PGD method by ensuring that the generated value is restricted between 0 and 1 using the $\operatorname{clip}_{[0,1]}$ operation. Furthermore, the function does not guide the patch δ towards any specific target label or bounding box.

The effectiveness of the AP attack is contingent upon several factors, analogous to the previously discussed attacks. Here, the ϵ value, or in this context, δ, determines the attack's strength. Yet, in this case, the size and location of the AP also play an integral role. An inconspicuously small or poorly located patch may evade detection by the DNN model.

The computational complexity of the AP attack significantly exceeds that of the PGD method, owing to its need for numerous transformations and complex optimization techniques when creating the adversarial example. Although this dissertation primarily aims to provide a clear visual understanding of the attacks, concealing the AP attack from human perception may prove challenging due to its distinct nature.

Despite these challenges, the AP attack concludes the series of attacks examined in this dissertation fittingly. Its inherent complexity, the unique implementation techniques that are mentioned in other attacks it employs, and the deviation from applying the attack to the entire image make it an effective demonstration of the varying methodologies that adversarial attacks can employ. As such, it serves to underscore the point that adversarial attacks are not bound by a single set of principles.

3.4 Adversarial Defenses

In devising a comprehensive defensive strategy against adversarial attacks on YOLOv5, it was essential to select a diverse array of defensive techniques. According to [35], adversarial defences can be split into two categories: *proactive* and *reactive*. Proactive defences typically involve modifications to the training process of the DNN to enhance its inherent robustness against adversarial perturbations. Conversely, reactive defences are designed to counteract adversarial attacks by applying specific alterations to the DNN's input, output, or architecture after the model has been trained and initialized. This section will explain the rationale and characteristics underpinning the selection of one proactive defence – AT, and two reactive defences – SS and FS. These defensive mechanisms have been chosen due to their demonstrated efficacy and prevalence in fortifying DNNs against adversarial attacks. Although these techniques share the common objective of enhancing the resilience of DNNs against adversarial attacks, they exhibit distinct properties that render them appropriate for this research. The specific differences, primarily concerning their implementation, computational complexity, and effectiveness, will be thoroughly explored in subsequent sections. Although these defences are frequently employed in unison in a single model, this dissertation seeks to portray the individual strengths and weaknesses of each technique in augmenting model robustness. This approach facilitates a nuanced understanding of the unique benefits and drawbacks of each defence. Consequently, each defence will be examined independently, offering readers a comprehensive insight into their respective functionalities. In summary, this section will provide an introduction to these three defences, exemplify their operational mechanisms, highlight their key differences, and discuss their limitations. This discussion will equip the reader with an understanding of what to anticipate in the forthcoming implementation and results sections.

3.4.1 Adversarial Training (AT)

AT is the initial defensive technique analyzed in this research. Renowned for its effectiveness in improving DNN robustness, its straightforward implementation and versatility make it a critical tool against various adversarial attacks. The operational mechanism is delineated in *Equation 7* of this paper [29].

$$h^* = \arg\min_{h \in H} \; \mathbb{E}_{(x, y_{\text{true}}) \sim D} \left[\max_{\|x_{\text{adv}} - x\|_\infty \leq \epsilon} L(h(x_{\text{adv}}), y_{\text{true}}) \right] \tag{7}$$

In this function, h^* represents the optimal model parameters, and $\arg\min_{h \in H}$ aims to find the lowest value of the ensuing expression for the given parameters. $\mathbb{E}_{[(x, y_{\text{true}}) \sim D]}$ signifies the expectation of the model's input concerning data distributions D, while $\max_{\|x^{\text{adv}} - x\|_\infty \leq \epsilon}$ represents the adversarial example x^{adv} that maximizes the loss function L. This function measures the deviation of the model's prediction on the adversarial example $h(x^{\text{adv}})$ from the true label y_{true}. The adversarial example is generated by adding a small perturbation (bounded by ϵ under the infinity norm) to the input x, thereby describing the worst-case scenario (the maxima covered in *Equation 5*). The objective is to identify adversarial examples that maximize the loss function and subsequently adapt the model's parameters to accommodate these adversarial inputs.

Despite its merits, AT has several limitations. As mentioned in [31], it can lead to reduced accuracy on non-adversarial images due to potential overtraining on adversarial images. This results in misclassifications when inference is applied to regular images. Furthermore, the high computational complexity of training a model on adversarial images can hamper the model's performance with inadequate hardware resources. Lastly, the effectiveness of AT largely depends on the type of attacks the model is trained to detect. For instance, a model trained solely on white-box sign gradient-based attacks may struggle to detect a black-box attack. Therefore, it is crucial that AT considers a broad spectrum of potential

adversarial attacks, has sufficient hardware resources, and maintains a balance between adversarial examples and regular image training to avoid significant accuracy losses.

AT is a valuable defence technique due to its easy implementation, scalability (relative to hardware), and potential to perform well against various adversarial attacks.

3.4.2 Feature Squeezing (FS)

The second defence technique under investigation is FS, a reactive approach that responds to the predisposition of adversarial attacks to exploit the unnecessarily large input feature space [34]. FS mitigates this vulnerability by reducing the degrees of freedom available to malicious inputs, thus '*squeezing*' the input feature space. The application of this technique for detecting adversarial examples is illustrated in *Figure 6*.

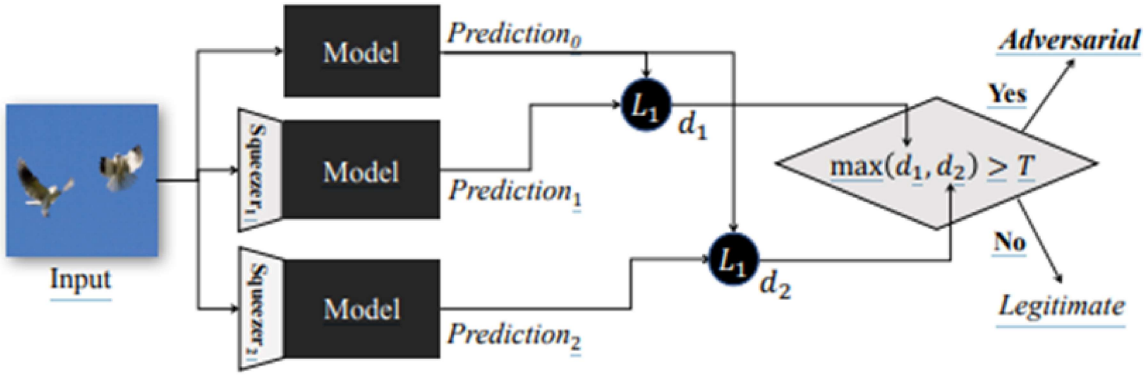


Figure 6: Feature squeezing as portrayed in the paper [34].

In this process, input is passed through a squeezer before being used for prediction. The discrepancy between the predictions based on squeezed and unsqueezed inputs is then calculated, and if this exceeds a specified threshold, the input is flagged as an adversarial attack.

However, in this research, FS is utilized as a preprocessing technique rather than a detection method. This implies the absence of a threshold for distinguishing adversarial inputs. Instead, the technique aims to modify or '*squeeze*' the inputs to mitigate potential adversarial attacks. This adaptation of FS comes with inherent limitations, as some adversarial attacks can be designed to use minimal image perturbations that can bypass the squeezing process [27]. Additionally, the balance between the extent of squeezing and maintaining the model's accuracy is critical. Over-squeezing might degrade the model's performance significantly while under-squeezing might leave the system vulnerable to attacks. Nevertheless, FS offers several advantages as a preprocessing technique, including low computational complexity, easy implementation, and provision of visual cues that signal its application. By carefully managing the trade-off between the extent of squeezing and model accuracy, this technique can potentially enhance the robustness of the system against adversarial attacks.

3.4.3 Spatial Smoothing (SS)

The final defence technique explored in this section is SS. Spatial Smoothing operates by blurring the input image, thereby reducing noise. This technique is performed as described in "*Digital Image Processing, 3rd edition*" [9], wherein each pixel in an image is replaced by the average intensity of its neighbouring pixels. These neighbouring pixels are defined by a kernel filter, a small matrix used for bullring, typically of dimensions *3x3* or *5x5*. The formula for SS is defined in *Equation 8*.

$$f'(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{8}$$

In this function, $\sum_{s=-a}^{a}$ represents the summation across horizontal pixels, while $\sum_{t=-b}^{b}$ denotes the summation across vertical pixels. The range for both s and t is from -1 to 1. The variable $w(s,t)$ stands for the weight assigned to the specific s and t pixels. Concurrently, the function $f(x+s, y+t)$ computes the new position of x and y pixels relative to s and t. The core principle of this function is the replacement of the original x and y pixels with the newly computed, weighted pixels, $w(s,t)$.

A noteworthy point from prior research [34] is that this type of formula is particularly effective for grayscale images, as it adeptly removes '*salt and pepper*' noise. However, due to the nature of the pixel replacement process, this technique has its limitations.

One significant limitation is the potential loss of information, particularly in areas of the image with sharp pixel intensity gradients. The averaging process can lead to the loss of critical image features. Another similarity it shares with FS is the potential for accuracy loss, underscoring the importance of judicious kernel selection during transformations.

Furthermore, while SS is highly effective for grayscale images, its application to full-colour images may result in substantial accuracy loss, potentially rendering the images unrecognizable. Despite this, I have opted to include this method in this study to demonstrate its potential capabilities with coloured images.

SS serves as an advantageous final line of defence due to its computational complexity and straightforward implementation. If the kernel size is chosen appropriately, the deployment of SS on coloured images could offer novel insights into its effectiveness in this relatively unexplored area.

3.5 Evaluation Metrics

In evaluating the attacks and defences, I will be using the same metrics as when evaluating the YOLOv5 model:

- **Precision**: The proportion of true positive predictions among all positive predictions. This measures the model's ability to correctly identify adversarial attacks. The formula: $\textit{Precision} = \frac{\text{True Positives}}{\text{(True Positives+False Positives)}}$
- **Recall**: The proportion of true positives among all actual positives. This indicates the model's ability to capture all possible attacks. The formula: $\textit{Recall} = \frac{\text{True Positives}}{\text{(True Positives+False Negatives)}}$
- **F1 Score**: The harmonic mean of *Precision* and *Recall*, providing a balanced measure of the model's performance. The formula: $\textit{F1} = \frac{2*\text{(Precision*Recall)}}{\text{(Precision+Recall)}}$
- **Precision-Recall Curves**: These will be used to visualize the trade-off between *Precision* and *Recall*.
- **Confusion Matrix**: Presented as a heatmap, this will give an overview of true and false positives and negatives.

In addition to these standardized metrics, I will conduct a qualitative evaluation. This will involve analyzing the visual difference between the original and adversarial images, as well as the interpretability of the defence-transformed images. These selected metrics should provide a comprehensive overview and insight into creating and evaluating adversarial examples with novel attack methods while measuring the robustness of a DNN model when applying the featured defensive techniques.

4 Implementation

In the following section, the implementation processes utilized in this study are outlined in detail. This commences with a discussion of the methods employed to process the selected KITTI dataset, detailing the specific techniques used to convert and preprocess images and labels respectively to fit the requirements of the YOLOv5 model. Subsequently, the procedures involved in the preparation and training of the YOLOv5n model are delineated. This includes a thorough explanation of the type of training conducted, as well as the selection and tuning of the model's parameters and hyperparameters, following the recommendations provided by the Ultralytics YOLOV5 GitHub page. Further to this, the implementation of the adversarial attacks, as discussed in the Methodology section, will be illustrated. Each attack—FGM, PGD, AP—will be accompanied by a pseudo-code to demonstrate the specific methods of implementation, alongside a comprehensive rationale for the parameters selected. In a similar vein, the defence techniques—AT, FS, and SS—will be explicated, including a detailed account of their operational mechanisms and the motivations behind the choices made during their implementation. Once these methods have been thoroughly addressed, the focus will shift to the testing procedures for these techniques. This discussion will emphasize the intent and specific goals of each test. Finally, the hardware and software utilized in the implementation phase will be addressed. This section will provide an overview of the technologies used, as well as a discussion of the challenges encountered during the implementation process and the solutions that were developed to overcome them.

4.1 Data

4.1.1 KITTI dataset

In preparation for training the YOLOv5n model, a crucial step was the preprocessing of the dataset. The KITTI dataset provided *7,481* labelled training images, which were exclusively employed for this phase. An initial task was to partition this training data into distinct subsets for training, testing, and validation. This partitioning followed a *70-20-10* distribution, respectively. The rationale behind this choice was to allocate a substantial majority of the images for training, while still reserving a portion for validation and a reasonably sized subset for testing to ensure the robustness of the model. The partitioning process involved randomly distributing the images and their corresponding labels among the training, validation, and testing directories. This random assignment ensures an unbiased selection of data. For ease of reference and analysis, the images were subsequently sorted. Subsequent to the partitioning of data, the labels required further processing to render them compatible with the YOLOv5 model. As outlined in *Table 1*, the labels provided by the KITTI dataset are extensive, including data that goes beyond the scope of this project which focuses solely on 2D object detection. Consequently, it was necessary to transform the labels into the YOLOv5 format, as illustrated in *Table 2*.

Index	Field	Description
0	Class Index	Category of the object (e.g., 0 for "Pedestrian")
1	x_center	X-coordinate of the center of the bounding box (normalized by the image width)
2	y_center	Y-coordinate of the center of the bounding box (normalized by the image height)
3	Width	Width of the bounding box (normalized by the image width)
4	Height	Height of the bounding box (normalized by the image height)

Table 2: YOLOv5 Label Format with Index

Algorithm 2 Convert KITTI labels to YOLOv5 format

procedure CONVERTKITTITOYOLO($labels\_dir$, $output\_dir$)
  Initialize $classes$ dictionary with class names as keys and indices as values
  for $filename$ in $labels\_dir$ **do**
    if $filename$ ends with '.txt' **then**
      $labels \leftarrow$ read labels from $filename$
      for $label$ in $labels$ **do**
        Split $label$ into components
        if class name is in $classes$ **then** extract class index, left, top, right, bottom from $label$
          Calculate center x = $(left + right)/2$
          Calculate center y = $(top + bottom)/2$
          Calculate normalized x, y, w, h
          if x or y or w or h > 1 **then**
            Print warning message
          end if
          Write class index, x, y, w, h to new file in $output\_dir$
        end if
      end for
    end if
  end for
end procedure

The process for this transformation is encapsulated in *Algorithm 2*. Essentially, this process involves normalizing the 2D bounding boxes and converting them to align with the YOLOv5 format. The object types were converted to indices ranging from 0 to 7, reflecting the order introduced in the *Methodology* section. These new indices were then normalized by dividing them by the image's width and height, as applicable. This process ensured the readiness of the images for input into the model.

4.1.2 Video

The evaluation of adversarial attacks and defences in real-time scenarios necessitates the use of a video, providing a more realistic environment than isolated images. To this end, a one-minute video capturing the streets of London was

acquired. In order to enable the application of adversarial attacks and defences on this video, it was necessary to convert the video into discrete frames. The process of extracting labels from the video frames poses certain challenges, which are addressed in detail in the subsequent '*Challenges and Solutions*' section of this document.

4.2 YOLOv5n training

In this section, I will describe the steps taken to train an optimal YOLOv5n model on a custom KITTI dataset. To train a YOLOv5n model on custom data, pre-trained weights were used and the model was fine-tuned for the specific task at hand. As recommended in [13], the '*yolov5n.pt*' pre-trained model weights on *COCO val2017* dataset [20] were used as a starting point for training. To initialize the model training, a "*KITTI.yaml*" file was created to provide a reference for the model's training and validation data and the objects it needed to detect. The model was trained for *300 epochs*, a value that was determined based on hardware availability and time constraints for this research paper. The model's architecture was based on the yolov5n model configurations. Hyperparameters for the model were set to *hyp.scratch-low*, *hyp.scratch-med*, and *hyp.scratch-high*, with each hyperparameter increasing in computational complexity and potential performance. To balance accuracy and computational expense, *hyp.scratch-med* was selected, with a slightly lower performance that would make adversarial methods more significant. For determining the anchors for the input, YOLOv5 *AutoAnchors* are utilized. The *AutoAnchors* find the best anchors, by analyzing the ground truth bounding boxes in the training data and using *K-means clustering* [11] to determine the best anchor box dimensions. This method automatically calculates the most suitable anchor box sizes for the dataset being used. The input image size was determined to be *1240x1240*, larger than the default input size of a YOLOv5n model, to improve model accuracy while keeping its original rectangle form. The model was trained without cropping images, and the final input is depicted in *Table 3*.

Parameter	**Value**
img_size	1240x1240
weights	yolov5n.pt
data	KITTI.yaml
cfg	yolov5n.yaml
hyp	hyp.scratch-med.yaml
rectangle	True

Table 3: Parameters for the train.py command

4.2.1 Evolving the hyperparameters

The process of parameter evolution is crucial in optimizing the model's parameters to enhance performance with custom data, a feature embedded within the YOLOv5 repository [13]. Initially, hyperparameters, identical to those used for the "$KITTI\_med$" dataset, are set. These are subsequently input into a fitness function, as shown in *Algorithm 3* [32].

Algorithm 3 Fitness function for YOLOv5 evolution

function FITNESS(x)
    Initialize w as $[0.0, 0.0, 0.1, 0.9]$ ▷ weights for [P, R, mAP@0.5, mAP@0.5:0.95]
    return $(x[:, :4] * w).sum(1)$
end function

The function calculates the weighted summation of *Precision*, *Recall*, mAP@0.5, and *mAP@0.5:0.95*, the first four components of input *x*, using the initialized weights *w*. Afterwards, parameters undergo mutation or evolution, as detailed in *Algorithm 4*.

Algorithm 4 Hyperparameter Mutation in YOLOv5 Evolution

procedure MUTATEHYPERPARAMETERS
  Initialize $mp, s \leftarrow 0.8, 0.2$ ▷ Mutation probability, sigma
  Initialize $g \leftarrow$ array of gains for each hyperparameter
  Initialize $v \leftarrow$ array of ones of size g
  while all elements of $v = 1$ **do** ▷ Mutate until a change occurs
    $v \leftarrow (g{*}(\text{RANDOM}(length(g)) < mp){*}\text{RANDOMNORMAL}(length(g)){*}\text{RANDOM}{*}s{+}1).clip(0.3, 3.0)$
  end while
  for each i, k in enumerate(hyperparameters) **do**
    hyperparameters[k] $\leftarrow x[i+7] * v[i]$ ▷ Mutate
  end for
  for each k, v in meta **do**
    hyperparameters[k] $\leftarrow \max(\text{hyperparameters}[k], v[1])$ ▷ Lower limit
    hyperparameters[k] $\leftarrow \min(\text{hyperparameters}[k], v[2])$ ▷ Upper limit
    hyperparameters[k] $\leftarrow \text{ROUND}(\text{hyperparameters}[k], 5)$ ▷ Significant digits
  end for
end procedure

Algorithm 4 describes parameter mutation or evolution for better alignment with custom data. The *mutation probability (mp)* is set to *0.8* (80% mutation probability), and the *mutation strength (s)* to *0.2* (20% mutation strength magnitude). This ratio enhances the likelihood of parameter mutation, thus improving performance. Next, arrays *g* and *v*, which represent the gains of each hyperparameter, are initialized.

The algorithm enters a loop where elements of *v* are probabilistically perturbed. This is achieved by adding random numbers, multiplied by *g* and *s*, to each element of the array, given that the random number is less than *mp*. The perturbed *v* is used to mutate the current hyperparameters *x*, updating each hyperparameter by multiplying it by the corresponding element of *v*.

Each mutated hyperparameter is constrained within its lower and upper limits and rounded to five significant digits for *precision*. Despite the effectiveness of this method, it has limitations. While the documentation recommends parameter evolution over *300 epochs*, the evolution was limited to *100 epochs* in this study to reduce training time while maintaining improved results.

After the implementation, *hyp_evolve* parameters are obtained and used to train a new model, following the same procedure as in YOLOv5 training. The key difference is the use of the *hyp_evolve* parameter instead of the *hyp.scracth_med* parameter. The new model, named $KITTI\_med\_evolve$, is used in generating adversarial attacks and defences.

4.3 Adversarial attacks

In this section, I will detail the process of creating adversarial examples using the *Adversarial-Robustness-Toolbox (ART)* [30], a Python library. I focused on generating adversarial attacks from images within the testing folder of the KITTI dataset, creating three distinct adversarial attacks for each image. The specific methods for each attack will be elaborated on in their respective sections, but the general pipeline for creating these adversarial images is outlined in *Algorithm 5*. This algorithm effectively establishes a consistent testing ground for evaluating the attacks by applying all three to each image. It also provides a clear demonstration of both the differences and similarities between the attacks. Detailed discussions of individual attack implementations will be presented in the following sections.

Algorithm 5 Adversarial Attack Generation

```
Load the trained model and set it to evaluation mode
Initialize the custom YOLOv5n model
Get the list of image filenames from the test image folder
for each image filename in the test image folder do
    Load and preprocess the image
    Create a copy of the image for the adversarial attack

    Initialize the FGM
    Generate the adversarial image using FGM

    Initialize the PGD
    Generate the adversarial image using PGD

    Predict the target for the image using the estimator
    Initialize the Adversarial Patch PyTorch (AP)
    Generate the adversarial patch using AP
    Apply the patch to the image

    Save the adversarial images to their respective output folders
end for
```

4.3.1 FGM Implementation

In this section, the implementation of the FGM using ART is discussed. This section elucidates the parameters required for the FGM attack and the rationale behind the selection of particular values for these parameters. The primary algorithmic structure of the ART FGM attack is represented in *Algorithm 6*.

Algorithm 6 FGM in ART

```
procedure FGM(x, y, mask, eps, eps_step, project, random_init)
    if random_init then
        x_adv ← x + random_perturbation(x, eps)
    else
        x_adv ← x
    end if
    for i in 1, ..., num_batches do
        batch_x ← x_adv[i]
        batch_y ← y[i]
        perturbation ← compute_perturbation(batch_x, batch_y, mask)
        x_adv[i] ← apply_perturbation(batch_x, perturbation, eps_step)
        if project then
            perturbation ← projection(x_adv[i] − x[i], eps)
            x_adv[i] ← x[i] + perturbation
        end if
    end for
    return x_adv
end procedure
```

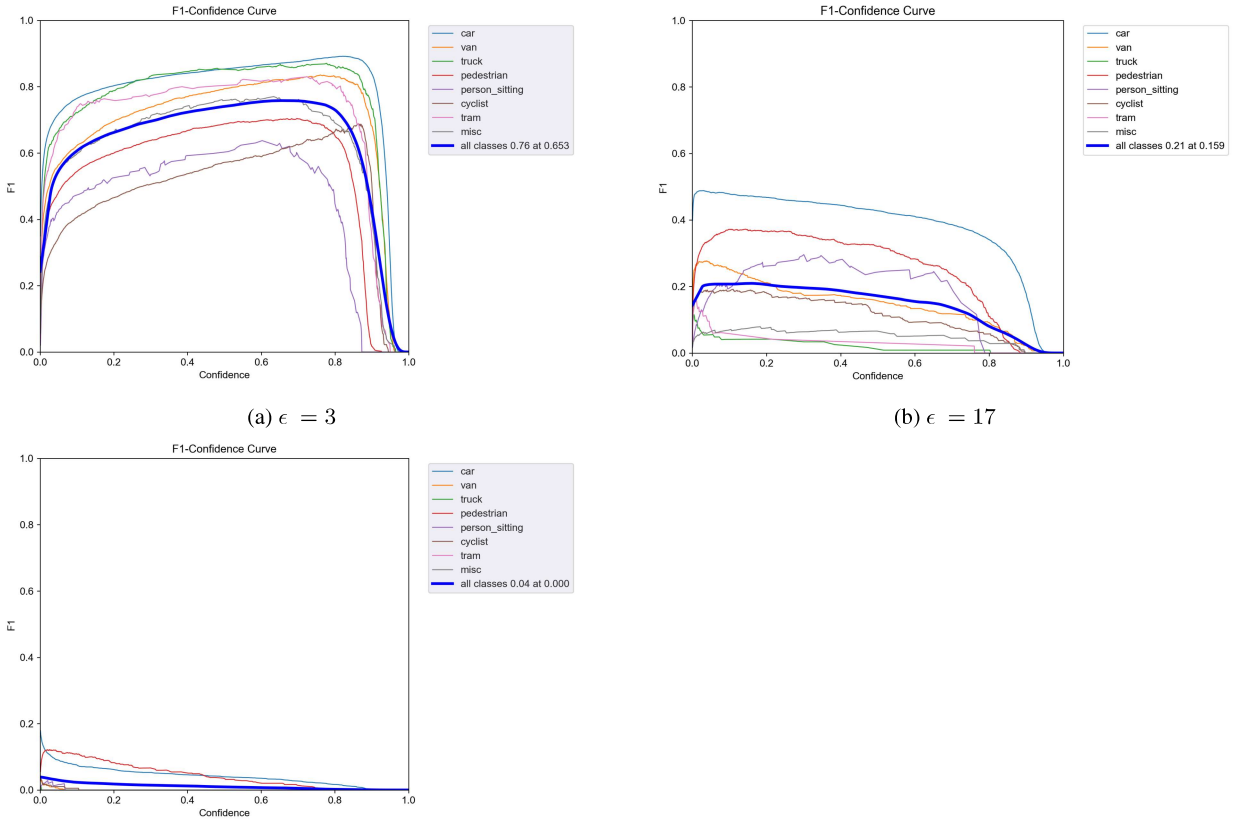
In this algorithm, x and y represent the image data and corresponding labels, respectively. If the option for random perturbations is enabled, these are added to the adversarial sample x_adv. If not, x_adv is set to the original input x. The algorithm then enters a loop iterating over the defined batches. The `compute_perturbation` function calculates the adversarial perturbation based on the loss gradient with respect to `batch_x` and `batch_y`, which represent the current data and labels, respectively.

Parameter	Explanation
estimator	The estimator object to use for crafting adversarial examples.
norm	The type of norm to use when measuring the size of the perturbation. Common choices include the L2 norm and the L-infinity norm.
eps	The maximum size of the perturbation, measured in the specified norm.
eps_step	The step size to use when updating the perturbation in each iteration of the attack.
targeted	Whether the attack is targeted or untargeted. In a targeted attack, the goal is to misclassify the input as a specific target class, while in an untargeted attack, the goal is simply to cause misclassification.
num_random_init	The number of random initializations to use when crafting adversarial examples.
batch_size	The number of images to process in each batch during adversarial training or testing.
minimal	Whether to compute the minimal adversarial perturbation that causes misclassification.
summary_writer	The SummaryWriter object to use for logging metrics during adversarial training.

Table 4: ART FGM attack parameters.

The parameters chosen for the FGM attack are displayed in *Table 4*. The estimator parameter is set to the trained *YOLOv5_med_evolve* model. The norm parameter, which determines the maximum allowable perturbation when executing the attack, is set to infinity '*inf*' (*L-infinity norm*). This selection was motivated by the desire to emphasize the importance of the epsilon values in the attack, as the norm constrains the perturbation of each pixel to the range [*-epsilon, epsilon*].

Three epsilon values were selected at random for comparison: a value in the range *[0,10)*, a value in the range *[10,20)*, and a value in the range *[20,40]*. The resultant images and *F1 scores* are displayed in *Figure 7*.



(a) $\epsilon = 3$

(b) $\epsilon = 17$

(c) $\epsilon = 34$

Figure 7: FGM attack with different ϵ values

Upon visual inspection of the generated images, it is clear that images created with $\epsilon = 3$ exhibit no discernible perturbations, which is reflected in their *F1 scores* being 0.76 at threshold 0.653. Conversely, images with $\epsilon = 17$ display visible perturbations and their *F1 score* of 0.21 at threshold 0.159 indicate successful deception of the model. Finally, images with $\epsilon = 34$ are highly perturbed to the point where object recognition is challenging with the lowest *F1 score* at 0.04 at threshold 0.000. This is reflected in the lowest *F1 scores* among the three epsilon values. Based on these results, $\epsilon = 17$ was selected for its ability to create noticeable yet not overly disruptive perturbations.

The targeted parameter is set to `False` as the selected attacks are not label-specific. The `num_random_init` parameter is set to 0 to maintain the simplicity of the attack, and the `batch_size` is set to 1 due to the one-image nature of the attack generation algorithm. The `minimal` parameter is left as `False`, as it is primarily applicable to classification-based adversarial attacks.

The chosen parameters are designed to generate adversarial examples with noticeable perturbations, but not to the extent that the images become unrecognizable. This approach ensures the creation of effective adversarial examples while maintaining the original context of the images.

4.3.2 PGD Implementation

In the subsequent section, the implementation of the PGD attack from the ART library is described. Refer to *Algorithm 7* for the detailed algorithm.
The algorithm accepts the image inputs x and y and iterates over the original image (x_orig), processed in batches of size $batch\_size$. Initially, the algorithm verifies if a random initialization (num_random_init) has been provided. If it is specified, random perturbations are introduced; otherwise, the adversarial example (x_adv) mirrors the input values.

Algorithm 7 Projected Gradient Descent attack in the ART library

```
procedure GENERATE(x, y)
    Log "Creating adversarial samples."
    for each x in x_orig, batched by batch_size do
        if num_random_init > 0 then
            Initialize random perturbation delta within the epsilon ball
            x_adv = x + delta
        else
            x_adv = x
        end if
        for i from 1 to max_iter do
            Compute gradient of the loss function w.r.t x_adv
            if targeted then
                x_adv = x_adv - eps_step * sign of gradient
            else
                x_adv = x_adv + eps_step * sign of gradient
            end if
            Clip x_adv to ensure it is within the epsilon ball around x
            Apply decay if applicable
        end for
    end for
    Return adversarial samples
end procedure
```

The iterative nature of the PGD method is observed in the nested for loop, where the algorithm iterates over x_adv up to the specified maximum iteration (max_iter), computing the loss value at each step. Depending on whether the method is *targeted*, the loss either gravitates towards the value of the specified class label (*targeted*) or is guided against the gradient for a universal attack (*non-targeted*).

Subsequently, x_adv is clipped to align with the epsilon's value limit (*epsilon-ball*), and, if specified, decay is applied. Upon completion of the loops, the manipulated image is saved as an adversarial example. *Table 5* presents the parameters utilized in the PGD attack.

Parameter	**Explanation**
estimator	The estimator object to use for crafting adversarial examples.
norm	The type of norm to use when measuring the size of the perturbation. Common choices include the L2 norm and the L-infinity norm.
eps	The maximum size of the perturbation, measured in the specified norm.
eps_step	The step size to use when updating the perturbation in each iteration of the attack.
max_iter	The maximum number of iterations to perform during the attack.
targeted	Whether the attack is targeted or untargeted. In a targeted attack, the goal is to misclassify the input as a specific target class, while in an untargeted attack, the goal is simply to cause misclassification.
num_random_init	The number of random initializations to use when crafting adversarial examples.
batch_size	The number of images to process in each batch during adversarial training or testing.
random_eps	Whether to sample the initial perturbation uniformly at random from the epsilon ball.

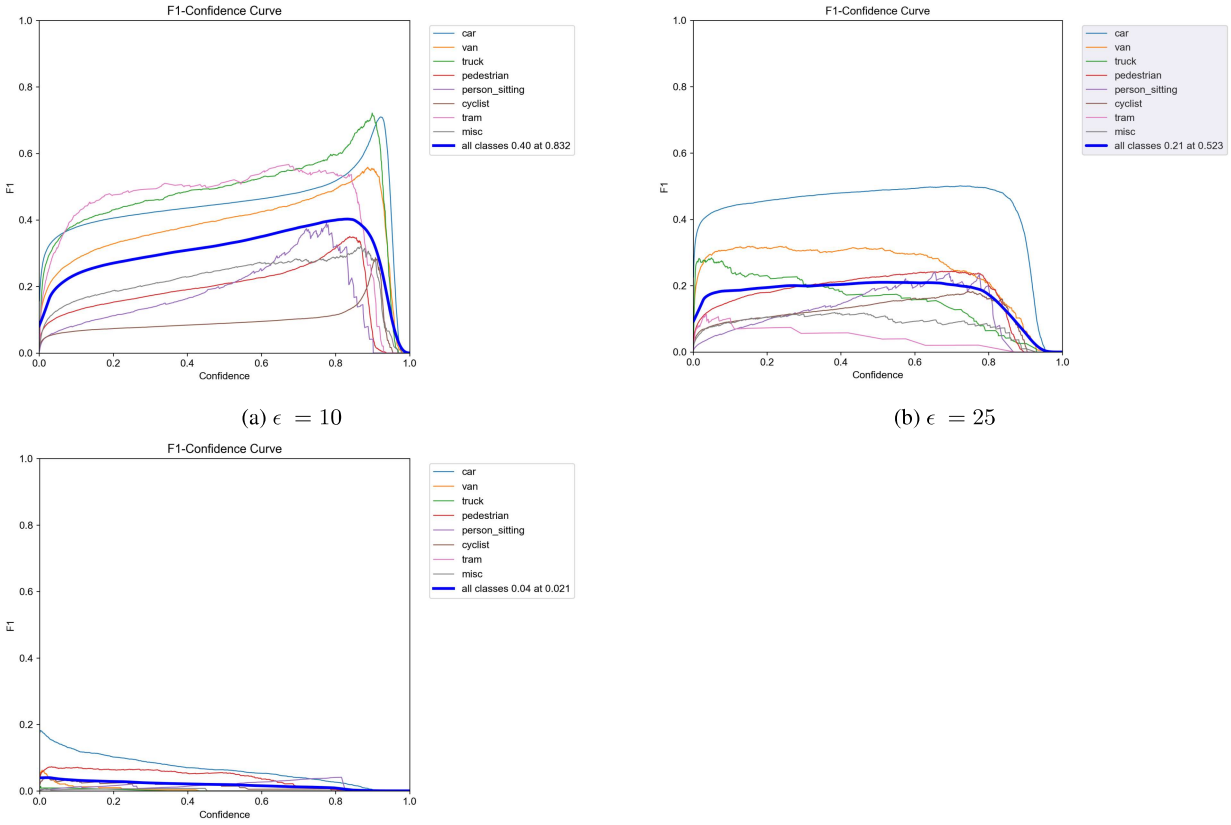
Table 5: ART PGD attack parameters.

Similar to the FGM section, the same values were adopted for *estimator*, *norm*, and *batch_size*. *targeted* and num_random_init were assigned *False* and *0*, respectively. The *random_eps* parameter was excluded from the attack, as the referenced study [17] did not test on PGD attacks, potentially leading to unanticipated outcomes.

The *eps* value was determined using the same principle as in the FGM attack, but given the iterative nature of the PGD method, wider ranges were defined for each value, with values randomly selected from their respective ranges.

Specifically, the first value was $\epsilon = 10$ (range: [0,20)), the second was $\epsilon = 25$ (range: [20,30)), and the third was $\epsilon = 48$ (range: [30,50)).

For experimental purposes, eps_step was set at 2 due to the sizable ranges, and max_iter was limited to *100*, based on findings from [22] that indicate *100 epochs* yield optimal outcomes in deceiving the model.



(a) $\epsilon = 10$

(b) $\epsilon = 25$

(c) $\epsilon = 48$

Figure 8: PGD attack with different ϵ values

As evident from *Figure 8*, the PGD attacks outperformed the FGM attacks. Even at the lowest value of $\epsilon = 10$, the perturbations were visible, and the model's accuracy was compromised as seen from the *F1 score* of *0.40* at threshold *0.832*. With $\epsilon = 25$, the perturbations were more pronounced, and the model's performance deteriorated further having the *F1 score* of *0.21* at threshold *0.523*. At $\epsilon = 48$, the perturbations were the most noticeable, significantly reducing the model's accuracy, *F1 score* being *0.04* at threshold *0.021* while obscuring certain image details.

After evaluation, $\epsilon = 25$ was selected, as it provides a balance between visible perturbations and preservation of image features while reducing the model's accuracy more than the previous method. The final PGD attack, with the specified parameters, effectively generates adversarial images that mislead the model without significant image distortion.

4.3.3 AP Implementation

In this section, I outline the implementation of an AP attack leveraging the ART library and discuss the selected parameters. The AP patch, referred to as *AdversarialPatchPytorch*, is an implementation of *RobustDPatch* [18] as clarified by the ART library developers[1]. *Algorithm 8* displays the pseudo-code employed in generating the attack in ART.

Algorithm 8 Adversarial Patch Generation in the ART library

```
procedure GENERATEADVERSARIALPATCH(x, y, kwargs)
    Initialize the parameters from kwargs (learning_rate, max_iter, batch_size, patch_shape, patch_location, patch_type, optimizer, targeted)
    Check and transform mask
    if y is None then
        Set y to the model's predictions on x
    end if
    Determine whether to use logits or probabilities based on estimator output
    Transform the labels into the appropriate format
    Create a PyTorch data loader based on input and labels
    for i from 1 to max_iter do
        for each batch in data loader do
            Transfer images and target to the appropriate device
            Apply patch to images in the batch
            Compute gradient of the loss function w.r.t the patch
            if targeted then
                Update the patch - learning_rate * sign of gradient
            else
                Update the patch + learning_rate * sign of gradient
            end if
            Clip patch to ensure it is within the specified limits
        end for
    end for
    Return the final adversarial patch
end procedure
```

Initially, the algorithm extracts and initializes parameters from `kwargs`. Subsequently, it verifies and transforms the mask, which is a binary replica of the image x, directing the algorithm where to position the patch in the image to deceive the model. If labels y are not provided, the model generates predictions based on x. The algorithm then determines whether to use logits (raw, unscaled output values produced by the model) or probabilities. Due to the YOLO architecture, logits are used in our case. Once the loader is created for x and y, the outer loop iterates over the image until reaching the maximum iteration limit, `max_iter`.

In the nested loop, each batch of images is transferred to either the CPU or GPU, and the patch is applied to the image. If it is a targeted attack, similar to the PGD implementation, the loss value is steered towards or away from the gradient. However, unlike PGD, the loss is also computed based on the specified `learning_rate`. Finally, the patch is clipped to the pre-determined dimensions. The parameters for AP are detailed in *Table 6*.

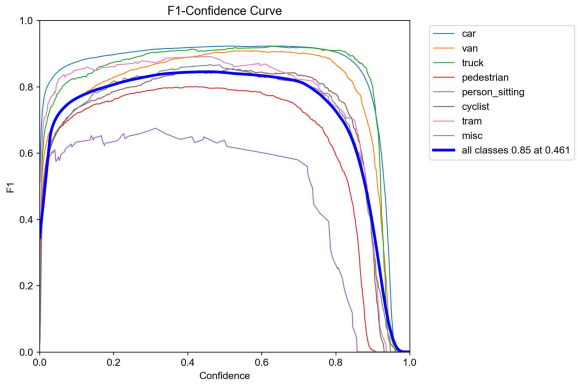
Parameter	**Explanation**
estimator	A trained estimator used for crafting adversarial examples.
rotation_max	The maximum rotation applied to the patch during the attack.
scale_min	The minimum scale to use when applying the adversarial patch.
scale_max	The maximum scale to use when applying the adversarial patch.
distortion_scale_max	The maximum distortion scale for the adversarial patch.
learning_rate	The learning rate of the optimization.
max_iter	The maximum number of iterations for the optimization.
batch_size	The number of images to process in each batch during adversarial training or testing.
patch_shape	The shape of the adversarial patch as a tuple of three integers.
patch_location	The location of the adversarial patch as a tuple of two integers.
patch_type	The type of the adversarial patch (either 'circle' or 'rectangle').
optimizer	The optimizer to use during the attack.
targeted	Whether the attack is targeted or untargeted. In a targeted attack, the goal is to misclassify the input as a specific target class, while in an untargeted attack, the goal is simply to cause misclassification.
summary_writer	The SummaryWriter object to use for logging metrics during adversarial training.
verbose	Show progress bars.

Table 6: ART Adversarial Patch PyTorch parameters.

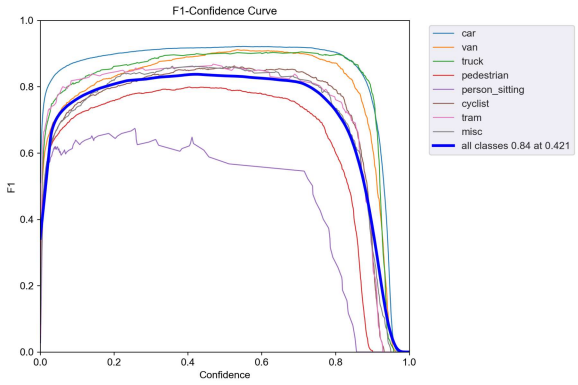
As previously noted, the $estimator$, $batch\_size$, and $targeted$ parameters remain consistent with prior methods. Despite numerous trials with different implementations for this method, the evaluation results exhibited minimal or no variation. The optimizer was the only constant across different parameter trials, excluding those previously mentioned. I chose PGD as the $optimizer$ due to its established methodology. *Table 7* displays the parameters for different iterations.

Table 7: Parameter Sets

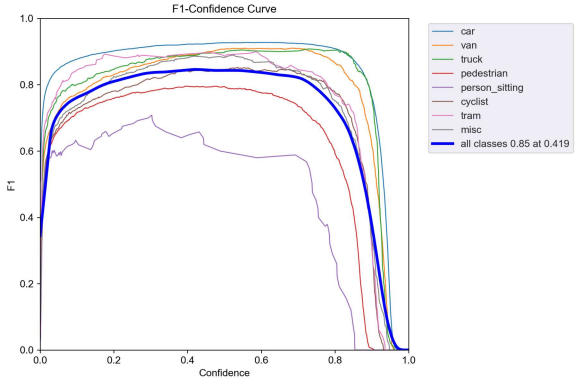
Parameters	Param 1	Param 2	Param 3
rotation_max	32.5	22.5	10.5
scale_min	0.1	0.4	0.4
scale_max	1.0	1.0	1.0
learning_rate	0.99	1.99	1.5
batch_size	1	1	1
max_iter	100	500	250
patch_shape	(3, 416, 416)	(3, 640, 640)	(3, 1248, 1248)



(a) Param 1



(b) Param 2



(c) Param 3

Figure 9: AP attack with different parameters

Figure 9 presents the *F1 scores*. The AP attack parameters seem to have minimal to no impact, as inferred from the *F1 scores*. The rationale behind this observation will be elaborated upon in the results section. With respect to AP generation, it is standard practice to train the attack over a significant number of epochs. Therefore, despite the parameters not causing substantial changes in the model, I opted to implement the attacks using $Param2$ values. In conclusion, even though the attacks may not pose a significant threat to the model, this research holds value in demonstrating how to enhance the attacks or assess the resilience of the YOLOv5 model against these types of attacks.

4.4 Adversarial defences

In this section, I discuss the implementation of defensive methods outlined in the methodology section, namely AT, FS, and SS. Unlike the attack methods, each defence has a distinct implementation and does not share a common algorithm. To elaborate, AT is trained similarly to $KITTI\_med\_evolve$, resulting in a new model, while FS and SS are implemented as preprocessing techniques for $KITTI\_med\_evolve$. The developed defences are specifically designed to counter and evaluate the previously discussed attacks.

4.4.1 AT Implementation

In this section, the implementation of the AT defence is discussed. The implementation process for this defence method is straightforward when compared to other attack or defence methods. The same techniques used in training the custom YOLOv5n model were applied here, with the exception that adversarial images and their corresponding default labels were used instead of images from the KITTI dataset. All images and labels were consolidated into one folder named $adv\_training$, which ultimately contained *4,489* files. Subsequently, a new model was trained using these images. The parameters used for this training process are outlined in *Table 8*.

Parameter	Value
img_size	1240x1240
weights	KITTI_med_evolve
data	adv_training
epochs	300
shape_of_input	rectangle

Table 8: Parameter values for the adversarial training implementation.

To facilitate a fair comparison with the default model, the parameters were largely kept consistent with those used in the $KITTI\_med\_evolve$ training, differing only in the data parameter. The final, newly trained model is referred to as $KITTI\_adv\_trained$.

4.4.2 FS Implementation

In this section, the implementation of FS defence is discussed. This implementation was based on the same approach as that of the adversarial attacks, using the ART library. The process of FS is demonstrated in *Algorithm 9*. Initially,

Algorithm 9 Feature Squeezing

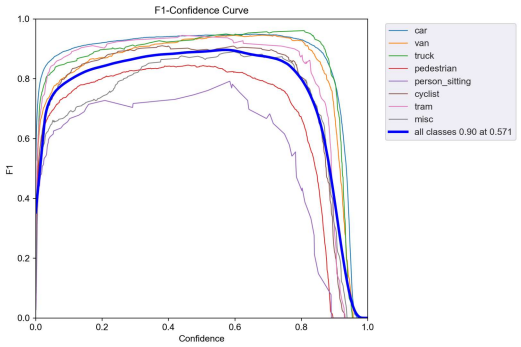
procedure FEATURESQUEEZING($x, clip\_values, bit\_depth$)
  $x\_normalized \leftarrow x - clip\_values[0]$
  $x\_normalized \leftarrow x\_normalized/(clip\_values[1] - clip\_values[0])$
  $max\_value \leftarrow \lfloor 2^{bit\_depth} - 1 \rfloor$
  $res \leftarrow \lfloor x\_normalized * max\_value \rfloor / max\_value$
  $res \leftarrow res * (clip\_values[1] - clip\_values[0])$
  $res \leftarrow res + clip\_values[0]$
  return res, y
end procedure

the input, x, is normalized by subtracting the lowest value of $clip\_value$ and then dividing the normalized value of x ($x\_normalized$) by the range of the $clip\_value$. Following this, the maximum value (max_value) for the transformations is computed using the equation 2^{bit_depth} - 1. The final transformed input (res) is then computed, which scales the data up to the range [0, max_value] while rounding to the nearest whole number. This process results in reducing the image's precision to the specified image depth. *Table 9* presents the parameters chosen for the FS implementation in ART.

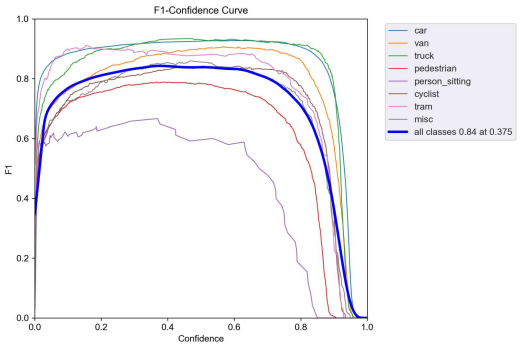
Parameter	Explanation
clip_values	A tuple containing the minimum and maximum values to clip the input data. The tuple should have the form (min, max), where min and max can be integers, floats, or ndarrays.
bit_depth	The bit depth to which the input data should be quantized.
apply_fit	A boolean indicating whether the feature squeezing transformation should be applied during the training (fitting) phase. The default value is False.
apply_predict	A boolean indicating whether the feature squeezing transformation should be applied during the prediction phase. The default value is True.

Table 9: Feature Squeezing parameters.

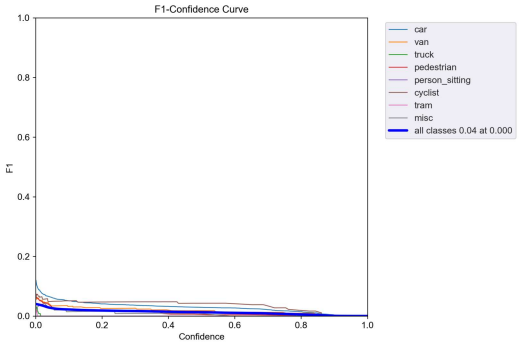
The $clip\_values$ were chosen in the range *[0,255]* to align with the $KITTI\_evolve\_med$ model, which also clips values within this range. The parameters $apply\_fit$ and $apply\_predict$ were retained in their default states, as they did not affect the final outcome of the attack. The selection of bit depth size bit_depth involved an experimental approach with two values from the range of *[1,8]*. The values chosen were $bit\_depth = 7$ and $bit\_depth = 4$. The rationale behind these choices was that *7* would represent the least minimum feature squeezing, and if the results with the value *4* proved inconclusive, smaller values could be explored. The outcomes of this experiment are illustrated in *Figure 10*.



(a) Default



(b) bit_depth=7



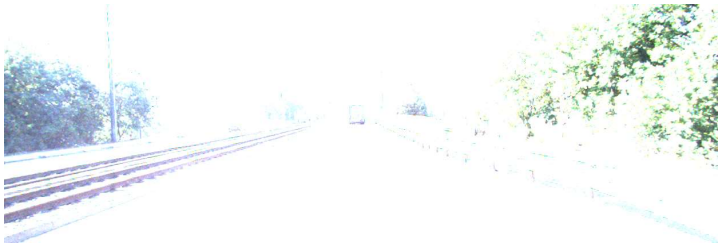
(c) bit_depth=4

Figure 10: FS defence with different bit_depth

The results in *Figure 10* and *11* revealed that images using $bit\_depth = 4$ were excessively squeezed, resulting in barely visible features, while their *F1 score*, compared to the default (*0.90 at 0.571*), is *0.04* at thresholds *0.000*. On the other hand, images created with $bit\_dept = 7$, while noticeably different, still maintained a certain level of accuracy. The *F1 score* of the default image was *0.84* at *0.375*. Given these results, a bit_depth value of *7* was chosen. While these parameters marginally decreased the accuracy of the original image, they may offer some mitigation for the attack.

(a) Default image

(b) image bit_depth=7



(c) image bit_depth=4

Figure 11: FS defence images with different bit_depth

4.4.3 SS Implementation

In this final section, the implementation of the SS defence method is discussed. For the purpose of this research, the SS implementation provided by the ART library was utilized. The pseudo-code for this algorithm is illustrated in *Algorithm 10*.

Algorithm 10 Spatial Smoothing

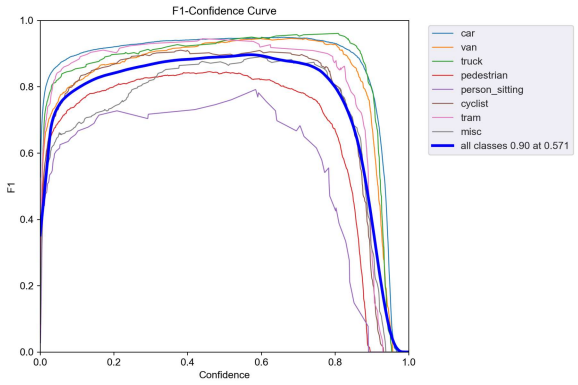
```
procedure SPATIALSMOOTHING(x, window_size, channels_first, clip_values)
    Determine filter size based on dimensions of x and channels_first
    result ← apply median filter to x with the determined filter size
    if clip_values is not None then
        Adjust the values in the result to fit within the range specified by clip_values
    end if
    return result
end procedure
```

Firstly, the dimensions of the filter size are determined considering the position of the colour channels denoted by $channels\_first$. Subsequently, the smoothed input result is calculated by applying a median filter of the determined $window\_size$ to the input x. Lastly, the result is transformed according to the specified image $clip\_values$, if provided. *Table 10* outlines the parameters for the ART SS implementation.

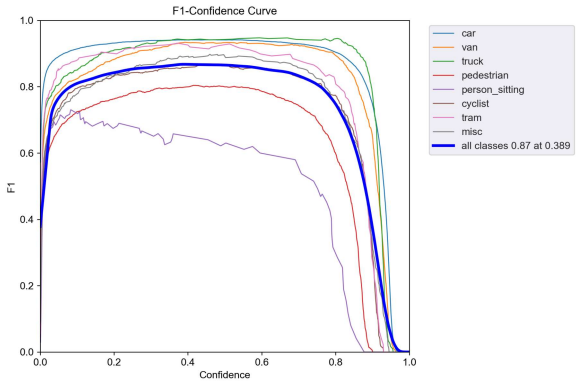
Parameter	Explanation
window_size	The size of the sliding window used in local spatial smoothing.
channels_first	A boolean value that indicates whether channels are the first dimension of the input data.
clip_values	A tuple representing the minimum and maximum values allowed for features. If not provided, no clipping is applied.
apply_fit	A boolean value indicating whether the defence should be applied during model training.
apply_predict	A boolean value indicating whether the defence should be applied during model prediction.

Table 10: ART Spatial Smoothing parameters.

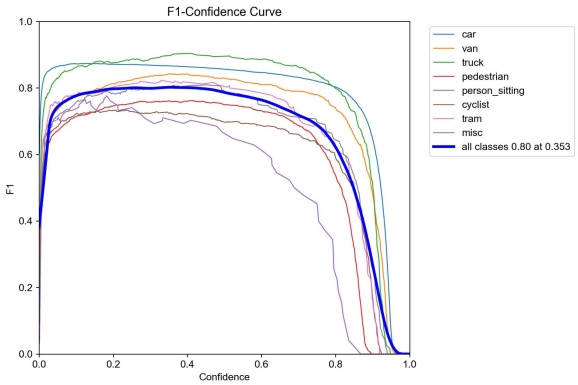
The parameters $channels\_first$, $clip\_values$, $apply\_fit$, and $apply\_predict$ were retained in their default states to ensure that the focus of the results remained primarily on the size of the Kernel filter, $window\_size$. The $window\_size$ must possess odd dimensions to ascertain that the kernel filter has a central pixel, thereby yielding better results. Therefore, the dimensions $window\_size$=3x3 and $window\_size$=5x5 were selected for testing. The outcomes of this experiment are depicted in *Figure 12* and *Figure 13*.



(a) Default



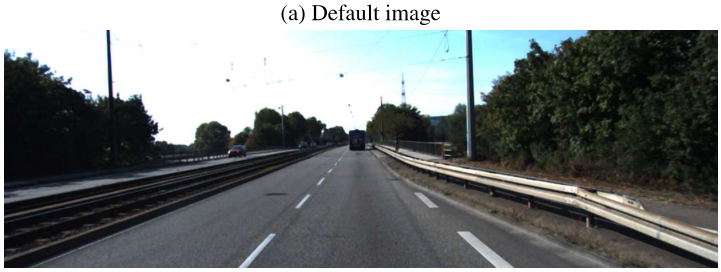
(b) window_size=3x3



(c) window_size=5x5

Figure 12: SS defence with different window_size

(a) Default image



(b) image window_size=3x3

(c) image window_size=5x5

Figure 13: SS defence images with different window_size

The results for $window\_size = 3x3$ indicate that images processed with this preprocessing technique exhibit fewer visible features while retaining the principal characteristics of the original image. Examining the *F1 score*, the *F1 score* for the changed input is *0.87* at thresholds *0.389*, while the default F1 is *0.90* at *0.571* threshold. The images generated with $window\_size = 5x5$ have more noticeable transformations but still retain key features making the image recognizable. But it does have a lower *F1 score* for the default image - *0.80* at a threshold of *0.353*. Given that the $5x5$ filter performed better in terms of preprocessing images, making transformations more visible, although having a slightly lower accuracy than the $3x3$ filter, it was selected for the implementation. The chosen parameters and the implementation generate images that have visible transformations, that differ from the other methods and potentially increase the robustness of the model.

4.5 Testing and Evaluations

The forthcoming section elucidates the methodologies employed for testing and evaluation throughout the course of this research. As mentioned in the previous sections, *F1 scores* were utilized as the prime metric for parameter optimization in both adversarial attacks and defence mechanisms. Upon achieving an optimized state of these operations, the evaluation phase was initiated. For evaluation purposes, two distinct datasets were employed. Primarily, the testing dataset derived from the KITTI collection was used to evaluate the performance of the $KITTI\_med\_evolve$ model under the prescribed settings. Subsequently, to simulate a real-world scenario and assess the model's performance in such a context, a video capturing the streets of London was utilized. The outcomes derived from these evaluations, presenting the performance metrics and insights into the model's behaviour under various conditions, are comprehensively discussed in the $Results$ section of this dissertation.

4.6 Hardware and Software

The specified hardware and software used in the implementation are portrayed in *Table 11*.

Hardware/Software	Version/Name	Explanation
YOLOv5	YOLOv5 v7.0	The version of the YOLOv5 object detection model used.
GPU	Nvidia GeForce GTX 1070 Ti	The specific GPU used for the computation.
Python	3.10.9	The Python version used in the environment.
ART	1.13.1	The version of the Adversarial Robustness Toolbox used.
KITTI	N/A	The KITTI dataset used for training and testing. No specific version.
CUDA	11.8	The version of CUDA used for GPU-accelerated computation.
torchvision	0.14.1	The version of the torchvision package, used for handling image data.
PyTorch	1.13.1	The version of the PyTorch deep learning framework used.

Table 11: Hardware and Software specifications.

For fast computational power a *Nvidia GeForce GTX 1070 Ti* GPU was used. $Python$ 3.10.9 was employed as the main programming language due to compatibility constraints with the ART and $YOLOv5$ libraries. Similar constraints influenced the choice of $PyTorch$ and $torchvision$ versions. ART was specifically utilized in version 1.13.1 rather than the newer 1.14.1, as the latter version encountered difficulties processing YOLOv5 predictions as labels in the AP implementation. Notwithstanding these constraints, the chosen set of hardware and software provides an optimized environment for the development and evaluation of object detection models, as well as their associated adversarial defences and attacks.

4.7 Challenges and Solutions

Throughout the implementation process, several challenges emerged. These challenges and their corresponding solutions are discussed in this section.

- **Computational Complexity:** The initial challenge encountered was the computational complexity associated with training a YOLOv5 model. The original plan was to utilize a local machine with a CPU for this task. However, due to the extensive duration required to train the model even with the lowest parameters, a more efficient strategy was necessary. The adopted solution was to leverage the powerful GPUs available in the Newcastle University gaming lab for the training process.

- **Integration of YOLOv5 Model and ART Library:** Another significant challenge was the integration of the YOLOv5 model with the ART library. To address this issue, assistance was sought from the developers of the ART library. They provided a helpful wrapper function for the model, as detailed in this GitHub discussion [26].

- **Adversarial Training Methodology:** The next challenge pertained to the AT method. The issue arose from the fact that the model was initially trained on adversarial examples generated from the KITTI test dataset. Consequently, this model could not be evaluated in the same manner as the models trained using other methods. To resolve this, new adversarial images were created using the same parameters, but these images were generated from the validation set of the KITTI dataset. This approach was successful because the model had only used the validation set for validation purposes and had not been initially trained on it.

- **The challenge in Label Acquisition from Video:** The process of evaluating the implemented attacks and defences on the custom model necessitated the conversion of the chosen video into image frames and the procurement of corresponding labels for each frame. To address this, I utilized the method outlined in *Algorithm 11*.

Algorithm 11 Creating labels using YOLOv5n model inference

```
procedure CREATELABELS(image, model)
    Load the image
    result ← Perform inference with model on the loaded image
    Initialize an empty list for labels
    for each detected object in the result do
        Get object class, confidence score, and bounding box coordinates
        label ← Create a label with object class, confidence score, and bounding box coordinates
        Add the label to the list of labels
    end for
    return the list of labels
end procedure
```

By employing predictions from the $KITTI\_med\_evolve$ model in the inference process, I was able to generate reasonably accurate labels suitable for testing the attacks and defences. One limitation of this approach is that it inhibits the model's evaluation in real-life scenarios using the same video, as it fails to simulate real-time detection. However, given its utility in facilitating the evaluation of other methods, it is still considered an effective solution.

5 Results

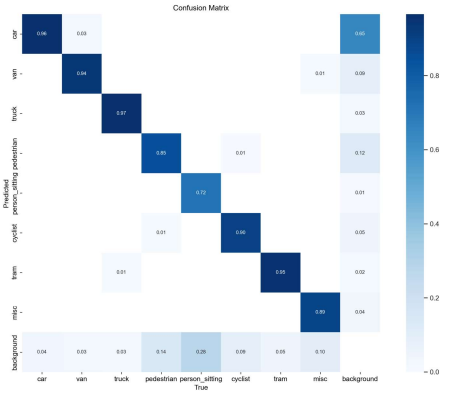
In the following section, the results derived from the model training, adversarial attacks, and defences will be meticulously analyzed and presented. While the custom models can't be evaluated on real-time video, as mentioned in the previous section, the attacks and defences will explore their effectiveness in different environments. Additionally, the various inputs generated by each method will be juxtaposed for a comparative evaluation.

5.1 Custom models

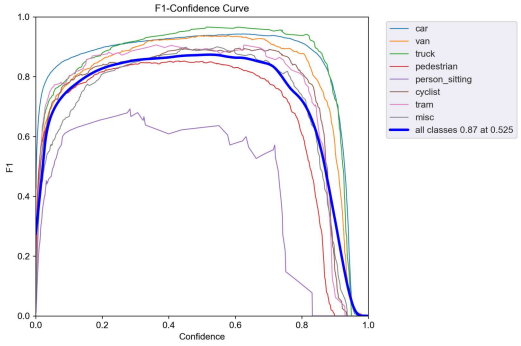
In this section, I will first evaluate the performance of the $KITTI\_med$ model and then compare and evaluate it against the $KITTI\_med\_evolve$ model. The purpose of this comparison is to assess the effectiveness of the model's evolution in terms of performance and accuracy. I will use the evaluation metrics established in the previous sections of the research paper.

5.1.1 KITTI_med

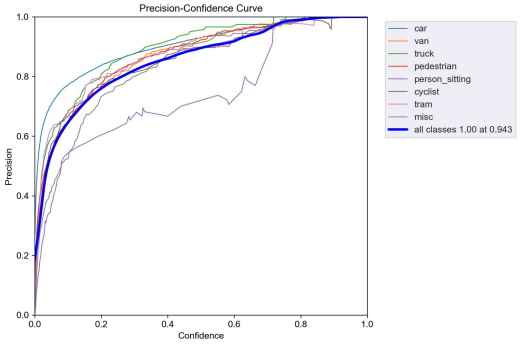
In this section, I will discuss the results from the training performed on the $KITTI\_med$ model, as described in the *YOLOv5n Training* section. *Figure 14* presents the performance metrics and the corresponding graphs.



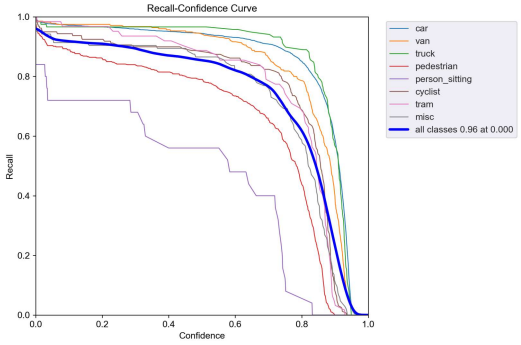
(a) Confusion matrix



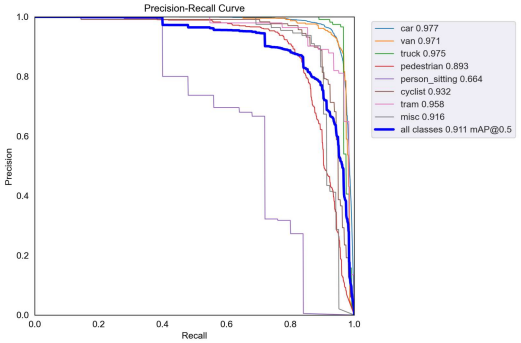
(b) F1 curve



(c) Precision curve



(d) Recall curve



(e) Precision-Recall curve

Metric	Score
F1	0.87 at 0.525
Precision	1.00 at 0.943
Recall	0.96 at 0.000
Precision-Recall mAP@0.5	0.911

(f) KITTI_med model performance results.

Figure 14: KITTI_med result metrics

As evident from the results, the default model demonstrates excellent performance in object detection, achieving high values for both *Recall* and *Precision* metrics. When looking at the confusion matrix, it shows that the model rarely misclassifies the object. The high *F1 score* of *0.87* at a threshold of *0.525* indicates a strong balance between *Precision* and *Recall*. Furthermore, the high *Precision-Recall score* suggests that the model effectively deals with data at varying thresholds. To determine if the model is underfitting or overfitting, I evaluated it on unseen data from the KITTI testing dataset. The results are shown in *Table 12*.

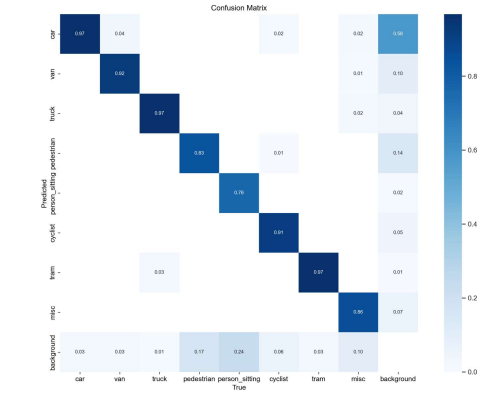
Metric	Score
F1	0.87 at 0.539
Precision	1.00 at 0.955
Recall	0.96 at 0.000
Precision-Recall mAP@0.5	0.906

Table 12: KITTI_med model performance results on the testing dataset.

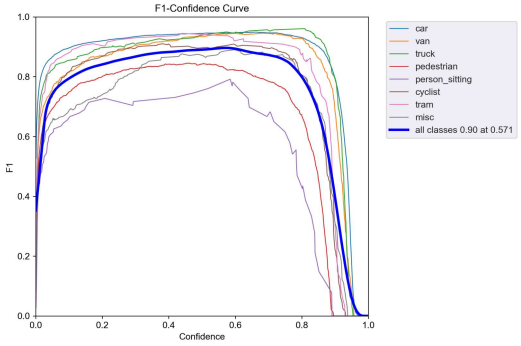
As the results reveal, the scores for each metric either match or vary by a small margin. Based on this, I can conclude that the model performs well when generalizing to unseen data and does not *overfit* or *underfit*. The model produced yields excellent results on custom data; however, minor improvements can be explored to determine if they lead to any changes in the model's performance, as discussed in the following section.

5.1.2 KITTI_med_evolve

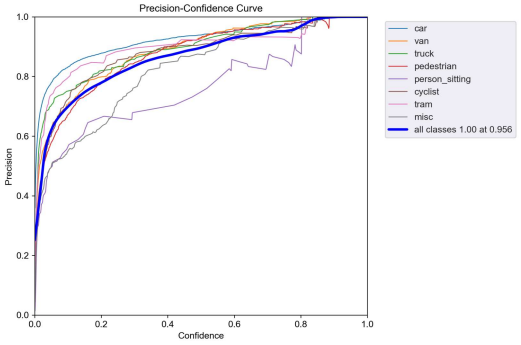
In this section, I will discuss and evaluate the results of the $KITTI\_med\_evolve$ model, which utilized the mutated hyperparameters presented in the Implementation section. *Figure 15* displays the model's training metrics and shows the corresponding plots.



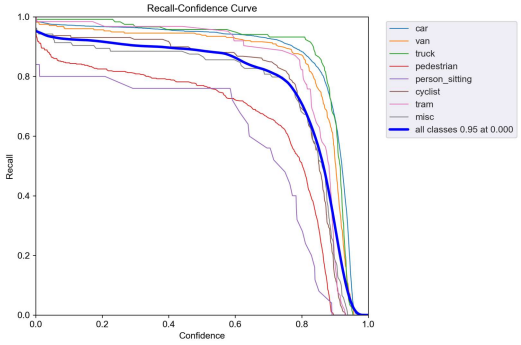
(a) Confusion matrix



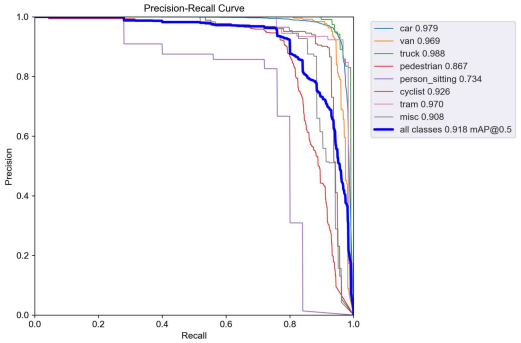
(b) F1 curve



(c) Precision curve



(d) Recall curve



(e) Precision-Recall curve

Metric	Score
F1	0.90 at 0.571
Precision	1.00 at 0.956
Recall	0.95 at 0.000
Precision-Recall mAP@0.5	0.918

(f) KITTI_med_evolve model performance results.

Figure 15: KITTI_med_evolve result metrics

Similar to the previous $KITTI\_med$ model, the $KITTI\_med\_evolve$ model demonstrates strong performance in detecting custom objects. Almost all metrics exhibit slight improvements, except for *Recall*, which remains at *0.95* at *0.000*, while the confusion matrixes are almost identical. Despite the small margin, the evolved hyperparameters do enhance the model's performance. To assess whether the model is overfitting or underfitting, I tested it on the same dataset used previously for the $KITTI\_med$ model. The results can be observed in *Table 13*.

Metric	Score
F1	0.88 at 0.416
Precision	1.00 at 0.964
Recall	0.96 at 0.000
Precision-Recall mAP@0.5	0.910

Table 13: KITTI_med_evolve performance results on the testing dataset.

As indicated in the table, none of the metrics shows a significant decrease, which implies that the model generalizes unseen data effectively and efficiently. The newly produced model demonstrates better accuracy and performance than the previous model.

5.1.3 Comparison

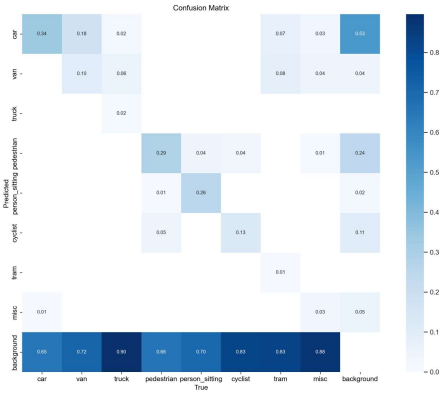
In reviewing the results of both $KITTI\_med$ and $KITTI\_med\_evolve$, it is evident that both models performed exceptionally well in detecting objects from the custom dataset. Moreover, both models effectively generalized unseen data. The evolved model demonstrated marginal improvements in most metrics, except for *Recall*. This modest improvement may be attributed to the limited number of epochs (*100*) during which the hyperparameter mutation was run, due to computational constraints. Increasing the number of epochs might potentially enhance the model's performance. Furthermore, it is worth noting that the KITTI dataset is relatively small in comparison to other popular datasets, such as *CIFAR100* or *ImageNet*. Consequently, the model may have already reached its optimal performance, and additional training might result in early stopping due to a lack of improvement or even lead to overfitting. In summary, the trained and evolved models exhibit satisfactory results and are appropriate for implementing attacks and defences.

5.2 Adversarial attacks

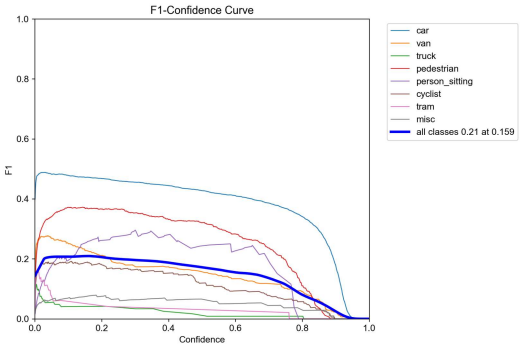
In this section, I will present and discuss the results obtained from the various attack methods used to generate adversarial examples. I will also compare their effectiveness in the context of the testing dataset and real-life video scenarios, and showcase the different adversarial images produced by each method. This analysis is crucial for understanding the vulnerability of the models to adversarial attacks and guiding the development of effective defence strategies.

5.2.1 FGM results

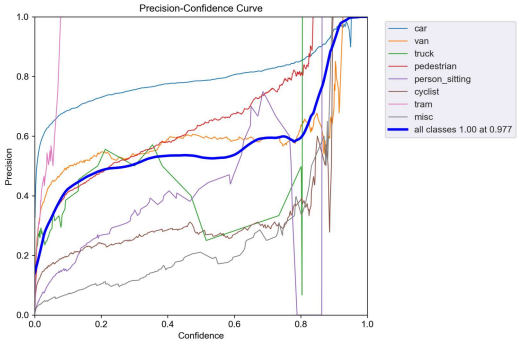
FGM Results on the Testing Dataset
For the first attack, FGM, I will first examine a list of metrics for the $KITTI_med_evolv$e model when evaluating it on the testing dataset. The metrics and the corresponding plots can be found in *Figure 16*.



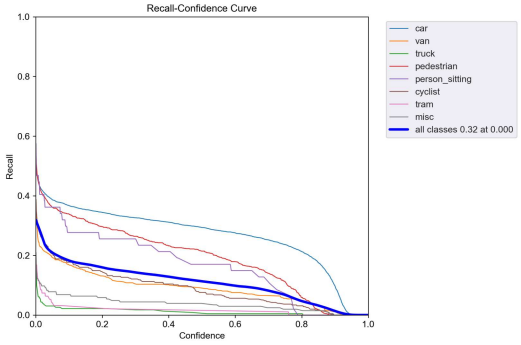
(a) Confusion matrix



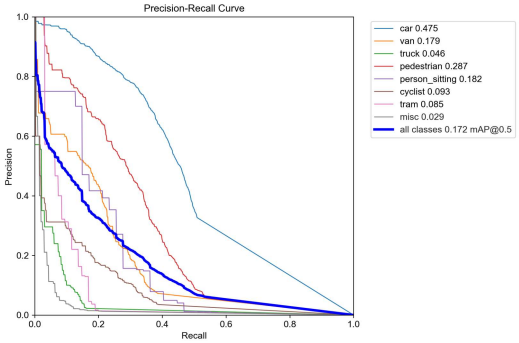
(b) F1 curve



(c) Precision curve



(d) Recall curve



(e) Precision-Recall curve

Metric	KITTI_med_evolve	FGM
F1	0.90 at 0.571	0.21 at 0.159
Precision	1.00 at 0.956	1.00 at 0.977
Recall	0.95 at 0.000	0.32 at 0.000
Precision-Recall mAP@0.5	0.918	0.172

(f) FGM attack's performance results

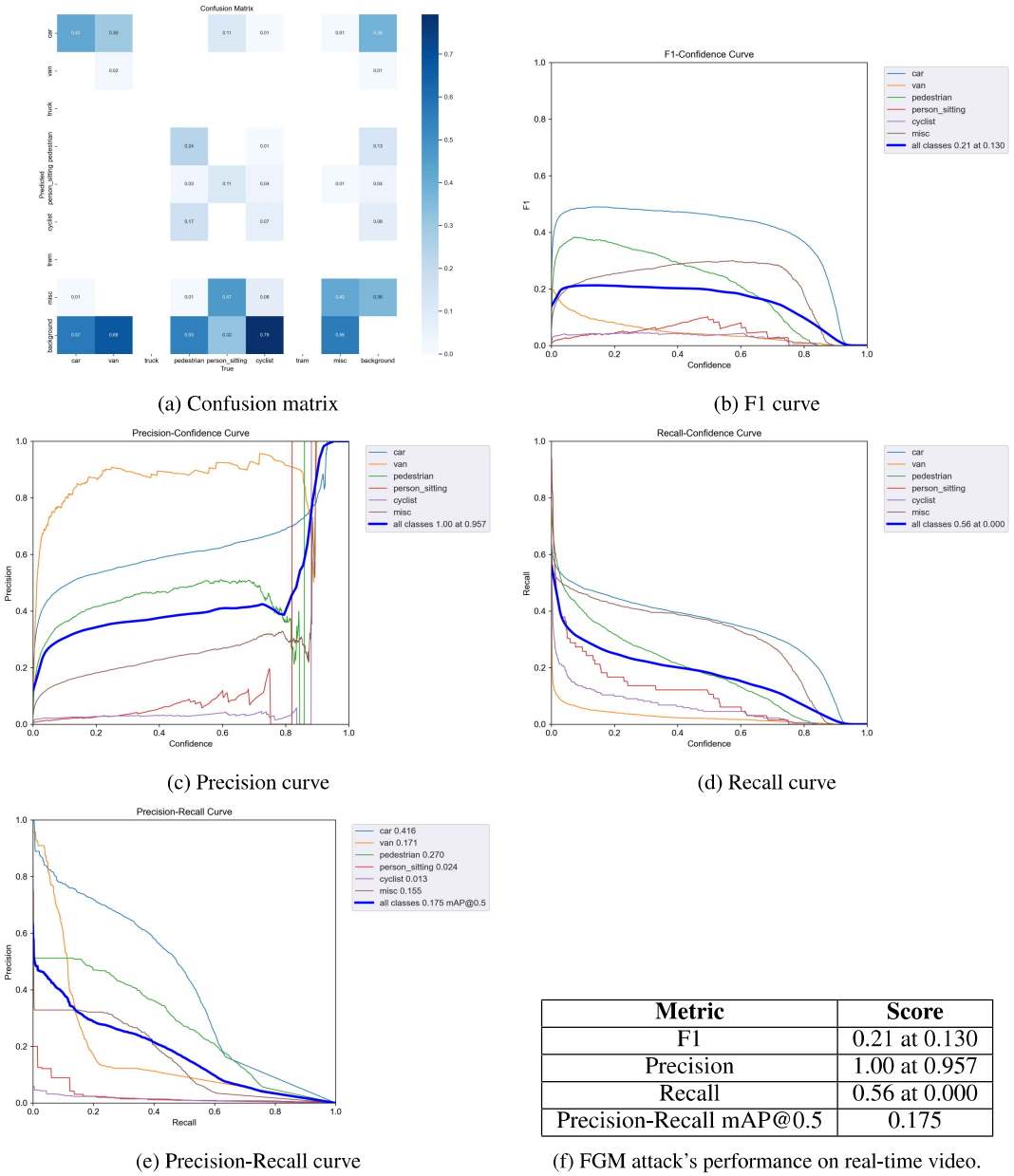
Figure 16: FGM testing result metrics

As can be seen from the metrics, the model's performance was drastically reduced due to the FGM attack. Examining the confusion matrix, it becomes evident that the model has a higher tendency to misclassify objects as background objects. One possible explanation for this observation is that background objects encompass a wide range of appearances, textures, and shapes, making them more challenging for the model to discriminate. This could lead to a greater number of False Negatives and a reduced recall, particularly when the model is subjected to adversarial attacks. The *F1 score* decreased to *0.21* at a threshold of *0.159*, while the *Recall score* dropped to *0.32* at *0.000*, and the *Precision-Recall score* reached *0.172*. Interestingly, the *Precision score* had an increase in its threshold to *0.977*, making it an outlier. This high *Precision score* could be attributed to the untargeted nature of the attack and the high epsilon value, which led to very noticeable perturbations. These perturbations did not increase the number of False Positive evaluations but rather increased the False Negatives, as indicated by the low *Recall score*. This observation supports my hypothesis that the high *Precision score* is due to the untargeted nature of the attack. Furthermore, we did not set a minimum threshold

to demonstrate the erratic predictions a model would make under adversarial attacks. Despite the high *Precision score*, the attack was successful in significantly degrading the model's performance and accuracy.

FGM Results on Real-life Video

This section discusses the results from the evaluation of the FGM method on real-time video. The metrics and the corresponding plots can be found in *Figure 17*.



(a) Confusion matrix

(b) F1 curve

(c) Precision curve

(d) Recall curve

(e) Precision-Recall curve

Metric	Score
F1	0.21 at 0.130
Precision	1.00 at 0.957
Recall	0.56 at 0.000
Precision-Recall mAP@0.5	0.175

(f) FGM attack's performance on real-time video.

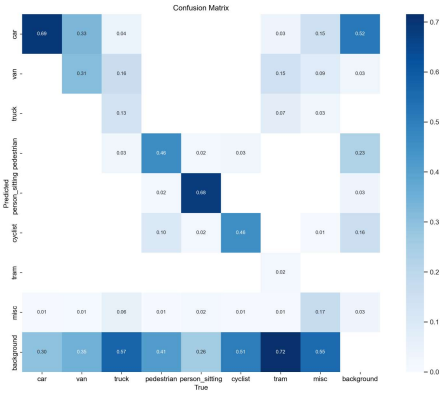
Figure 17: FGM video result metrics

Although there are no results for the $KITTI\_med\_evolve$ model due to the previously mentioned reasons, we can still observe that the attack is effective. The model's confusion matrix is more varied, but still favours the background class during classification. The *F1 score* remained low, at the same value, with a slight increase in threshold to *0.130*. The only notable difference is that the *Precision score's* threshold decreased to *0.957*, the *Recall score* increased to *0.52*, and the *Precision-Recall score* increased by *0.001* to *0.175*. The respective decrease and increase in *Precision* and *Recall*

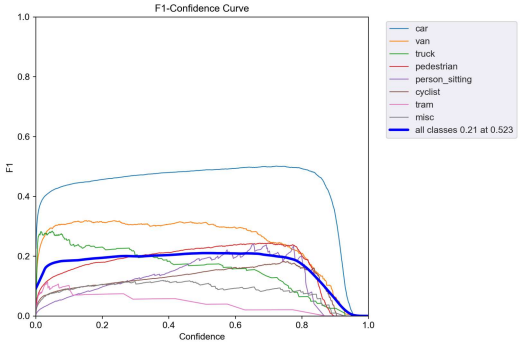
scores are likely due to the nature of the video. When converting the video to image frames, there is a high degree of spatial redundancy, as the images are overlapping. Additionally, the motion of objects and occlusion can affect the given parameters. Despite these factors, the results demonstrate that the attack remains effective during live testing.

5.2.2 PGD results

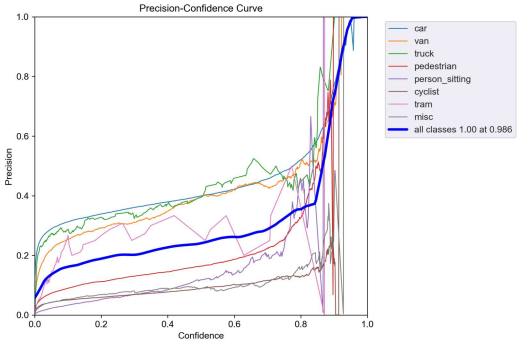
PGD Results on the Testing Dataset
The results and metric plots of the PGD attack on the testing dataset are shown in *Figure 18*.



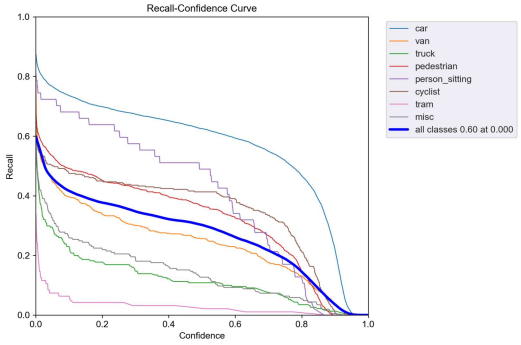
(a) Confusion matrix



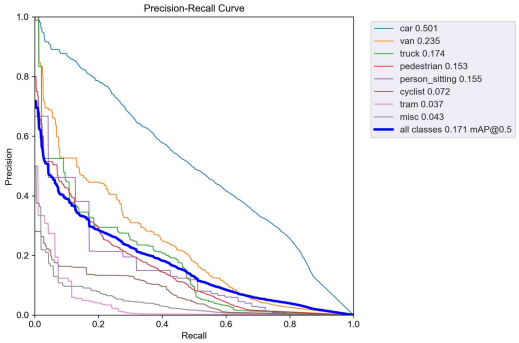
(b) F1 curve



(c) Precision curve



(d) Recall curve



(e) Precision-Recall curve

Metric	KITTI_med_evolve	FGM
F1	0.90 at 0.571	0.21 at 0.523
Precision	1.00 at 0.956	1.00 at 0.986
Recall	0.95 at 0.000	0.60 at 0.000
Precision-Recall mAP@0.5	0.918	0.171

(f) PGD attack's performance results

Figure 18: PGD testing result metrics

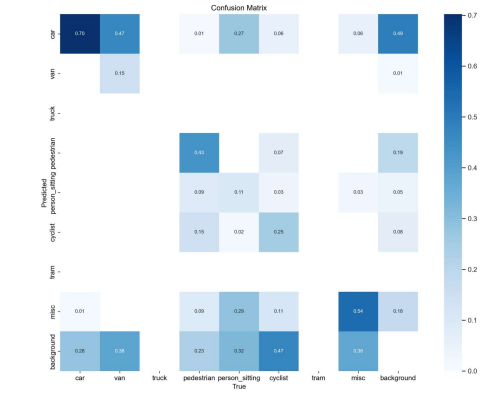
From the results, it is evident that the PGD attack significantly reduced the model's robustness, as demonstrated by the similarity of the confusion matrix to that of the FGM attack. The model exhibits a higher likelihood of misclassifying images as background objects.

The *F1 score* experienced a considerable decline to *0.21* at a *0.523* threshold, while the *Precision score* remained high at *1.00* at a *0.986* threshold. The *Recall score* also decreased to *0.60* at *0.000*, and the *Precision-Recall score* was registered at *0.171*. It is important to note that the PGD attack shares similar tendencies with the FGM attack regarding the *Precision* and *Recall scores*, which can be attributed to the reasons provided in the previous section.

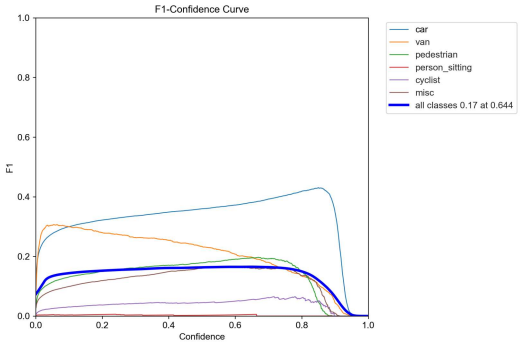
Although the resulting metrics for the PGD attack are generally higher than those of the FGM attack, with the exception of the PGD's *Precision score*, the attack is still regarded as a successful adversarial example. This evaluation contributes to a comprehensive understanding of the PGD attack's impact on the targeted model and highlights its effectiveness as an adversarial attack in comparison to the FGM attack.

PGD Results on Real-life Video

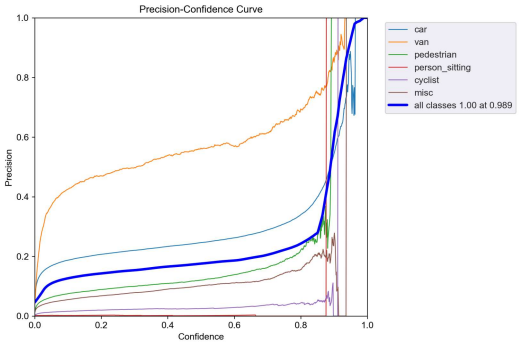
The following *Figure 19*, shows the results for PGD on real-time video.



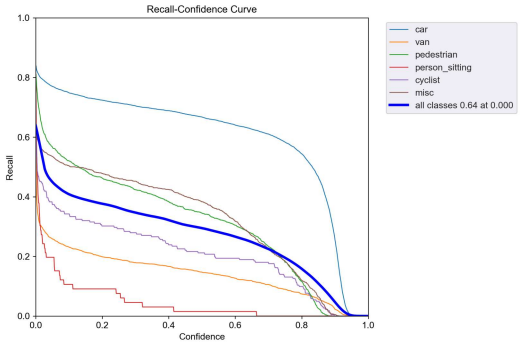
(a) Confusion matrix



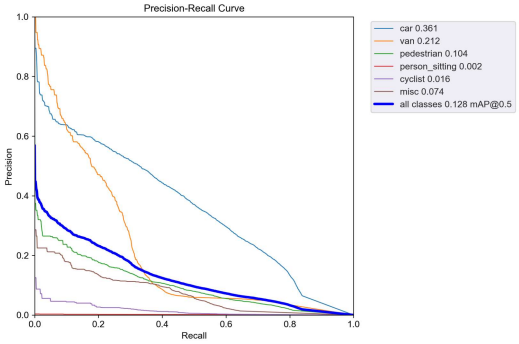
(b) F1 curve



(c) Precision curve



(d) Recall curve



(e) Precision-Recall curve

Metric	Score
F1	0.17 at 0.644
Precision	1.00 at 0.989
Recall	0.64 at 0.000
Precision-Recall mAP@0.5	0.128

(f) PGD attack's performance on real-time video.

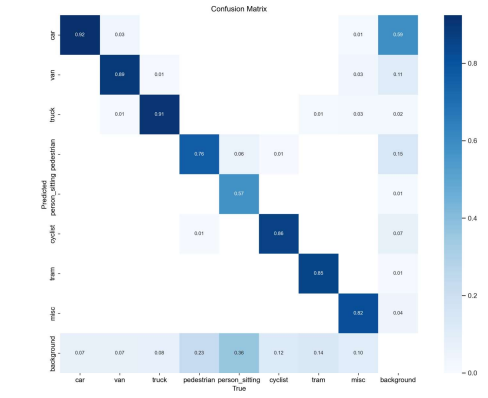
Figure 19: PGD video result metrics

The attack maintains its effectiveness in real-time environments, as evidenced by the confusion matrix, which is notably similar to that of the FGM attack. The matrix shows the misclassification of objects in various classes, with a higher inclination toward the background class. The *F1 score* remained low at *0.17* at a *0.644* threshold, while the *Precision* and *Recall scores* stayed within the same relative range, with values of *1.00* at *0.989* and *0.64* at *0.000*, respectively. The *Precision-Recall score* increased to *0.128*. Contrary to the FGM attack, there was no apparent disparity between the testing dataset results and the real-time video results. This can be attributed to the optimization-based algorithm employed in the PGD attack, as it underwent multiple optimization steps in both environments, resulting in similar outcomes. In conclusion, the PGD attack demonstrated satisfactory performance when evaluated in both scenarios, further emphasizing its effectiveness as an adversarial attack in comparison to the FGM attack in real-time environments.

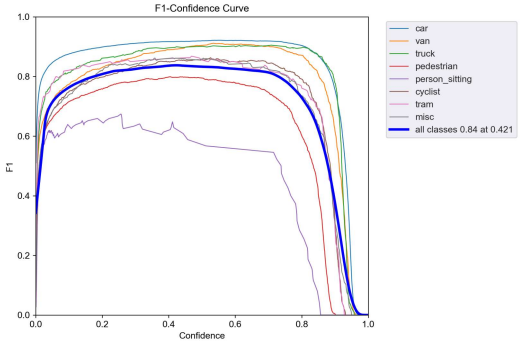
5.2.3 AP results

AP Results on the Testing Dataset

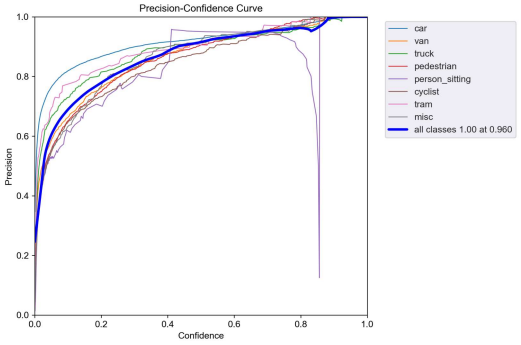
Figure 20 showcases the results and plots generated by the final attack, the AP attack.



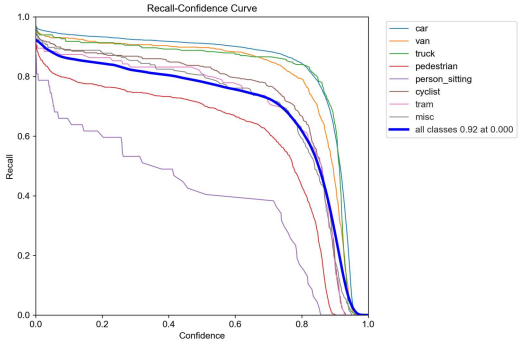
(a) Confusion matrix



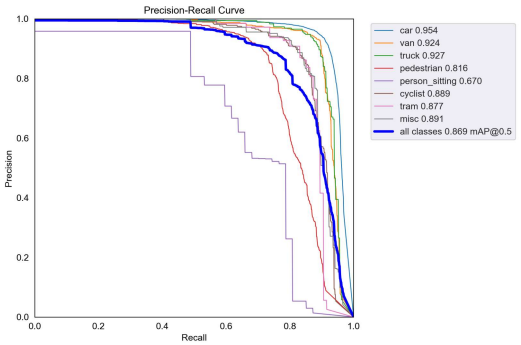
(b) F1 curve



(c) Precision curve



(d) Recall curve



(e) Precision-Recall curve

Metric	KITTI_med_evolve	FGM
F1	0.90 at 0.571	0.84 at 0.421
Precision	1.00 at 0.956	1.00 at 0.960
Recall	0.95 at 0.000	0.92 at 0.000
Precision-Recall mAP@0.5	0.918	0.869

(f) AP attack's performance results

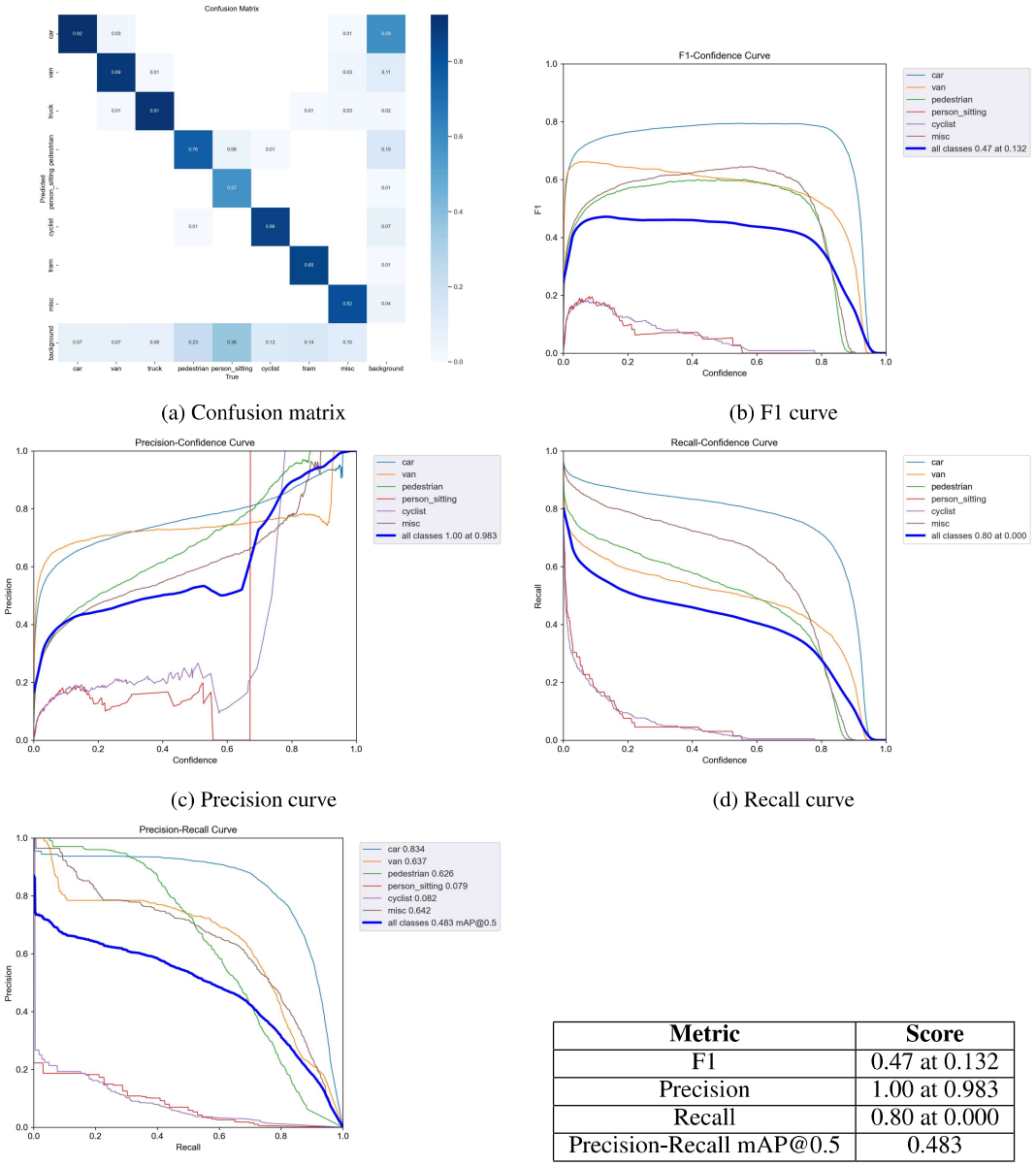
Figure 20: AP testing result metrics

Upon analyzing the results, it is evident that the AP attack is relatively ineffective in deceiving the model. The confusion matrix reveals a pattern similar to that of $KITTI\_med\_evolve$, with slightly higher values along the "*Background*" Y-axis. The other metrics exhibit only minor changes due to the attack, as follows: *F1 score* at *0.84* with a *0.421* threshold, *Precision score* at *1.00* with a *0.960* threshold, *Recall score* at *0.92* with a *0.000* threshold, and *Precision-Recall score* at *0.869*. These findings suggest that the model has not been significantly impacted by the attack, and the observed changes can be attributed to the attack's occlusion of objects. A potential explanation for the AP attack's limited effectiveness can be found in the research paper [21]. The paper reports favourable results when the model is

trained for *100,000* to *200,000* epochs. However, in this case, the attacks were trained for only *500 epochs*. This limited training duration may have drastically reduced the effectiveness of the attack.

AP Results on the Real-life Video

In evaluating the performance of the AP attack on real-life data, it is reasonable to expect similar results to those observed in the previous section. The results are displayed in *Figure 21*.



(a) Confusion matrix

(b) F1 curve

(c) Precision curve

(d) Recall curve

(e) Precision-Recall curve

Metric	Score
F1	0.47 at 0.132
Precision	1.00 at 0.983
Recall	0.80 at 0.000
Precision-Recall mAP@0.5	0.483

(f) AP attack's performance on real-time video.

Figure 21: AP video result metrics

Although the confusion matrix exhibits greater disparity in misclassification, the other metrics indicate that this is likely due to the nature of the data. The *F1 score* has decreased to *0.47* at a *0.132* threshold. However, the *Precision score* remains high at *1.00* with a *0.983* threshold, the *Recall score* at *0.80* with a *0.000* threshold, and the *Precision-Recall score* at *0.483*. These scores are significantly higher than those observed for the other attack methods, suggesting that the AP attack was not successful in compromising the model's robustness for the reasons discussed earlier. While

the AP attack did not effectively reduce the model's robustness, the generated adversarial examples still serve as an illustrative demonstration of the potential variety of adversarial attacks.

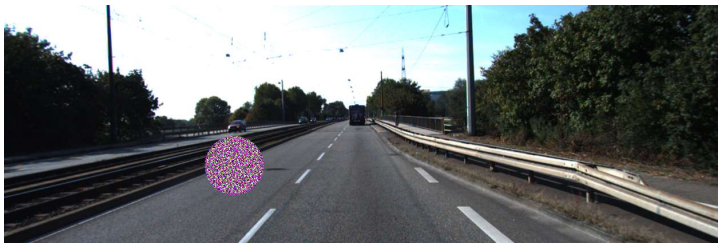
5.2.4 Attack visualization

In this section, the differences and similarities between the generated adversarial examples from various attack methods are examined in relation to the original image. The examples can be viewed in *Figure 22*.

(a) Default image

(b) FGM attack

(c) PGD attack



(d) AP attack

Figure 22: Images generated by the attacks.

An analysis of the generated examples reveals that each adversarial example exhibits distinct types of perturbations. While the objective was to create visible adversarial examples for the reader, the examples display varying noticeable features. Among the three, the AP-generated example is the most conspicuous, featuring a large, uncharacteristic patch in the image. In contrast, the FGM-created adversarial perturbations are more visible than those of the PGD, attributable to its straightforward algorithm, which trades perturbational complexity for ease and speed of implementation. Taking into account both the metric results and the generated examples, it can be concluded that the **PGD-created examples are the most effective**.

5.2.5 Summary

The summary of the results for each attack is presented in *Tables 14* and *15*.

Metric	KITTI_med_evolve	FGM	PGD	AP
F1	0.90 at 0.571	**0.21 at 0.159**	0.21 at 0.523	0.84 at 0.421
Precision	1.00 at 0.956	1.00 at 0.977	1.00 at 0.986	**1.00 at 0.960**
Recall	0.95 at 0.000	**0.32 at 0.000**	0.60 at 0.000	0.92 at 0.000
Precision-Recall mAP@0.5	0.918	0.172	**0.171**	0.869

Table 14: Comparison of FGM, PGD, and AP attack's performance results.

Metric	FGM	PGD	AP
F1	0.21 at 0.130	**0.17 at 0.644**	0.47 at 0.132
Precision	**1.00 at 0.957**	1.00 at 0.989	1.00 at 0.983
Recall	**0.56 at 0.000**	0.64 at 0.000	0.80 at 0.000
Precision-Recall mAP@0.5	0.175	**0.128**	0.483

Table 15: Comparison of FGM, PGD, and AP attack's performance on real-time video.

Table 14 provides an overview of the results obtained from evaluating the attacks using the testing dataset. It is evident that almost all attacks, with the exception of AP, reduce the effectiveness of the model, while not affecting *Precision* for the reasons previously discussed. Among the evaluated attacks, **FGM demonstrates the best performance in the testing dataset**, followed by PGD with a marginally higher *Recall score*. Conversely, the AP attack fails to deceive the model.

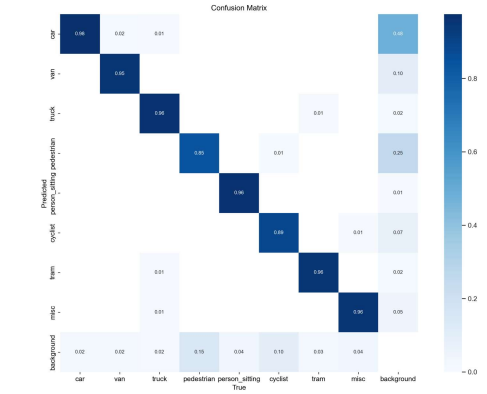
Table 15 reveals the performance of the attacks in real-life video scenarios. Both FGM and PGD prove to be effective in this context, while AP, consistent with the previous results, does not exhibit any signs of deceiving the model. Notably, **PGD performs better than FGM in real-life scenarios**, with FGM only displaying a lower *Recall score* and a decreased *Precision* threshold. The results suggest that FGM and PGD attacks successfully reduce the robustness of the model across both environments, whereas the AP attack has a minimal impact on the model's performance.

5.3 Adversarial defences

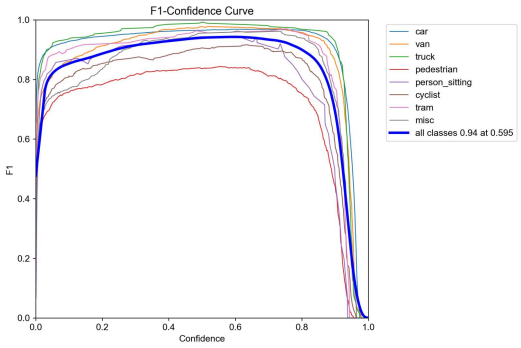
In this section, the effectiveness of the defensive methods in enhancing the custom model's robustness will be assessed in both testing and real-life environments.

5.3.1 AT results

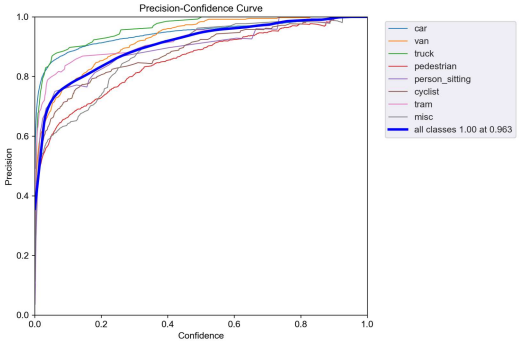
The first adversarial defence, AT, does not modify the input. Instead, it is a custom model designed to handle adversarial attacks. *Figure 23* presents the results of the model after training, compared to the $KITTI\_med\_evolve$, and its associated plots.



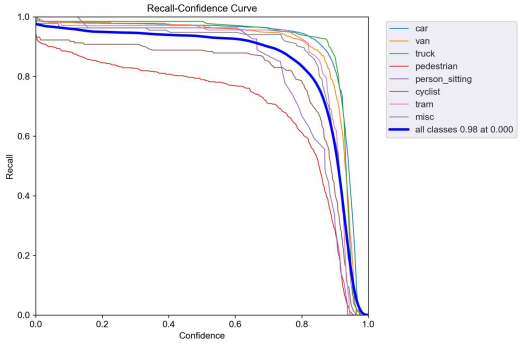
(a) Confusion matrix



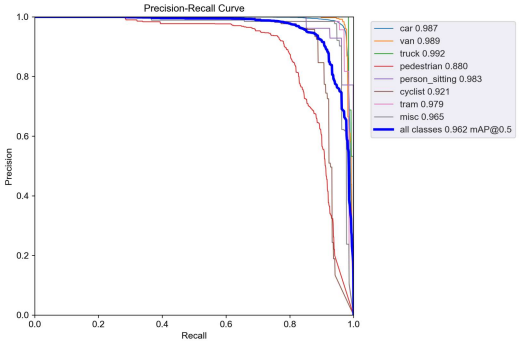
(b) F1 curve



(c) Precision curve



(d) Recall curve



(e) Precision-Recall curve

Metric	KITTI_med_evolve	AT
F1	0.90 at 0.571	0.94 at 0.595
Precision	1.00 at 0.956	1.00 at 0.963
Recall	0.95 at 0.000	0.98 at 0.000
Precision-Recall mAP@0.5	0.918	0.962

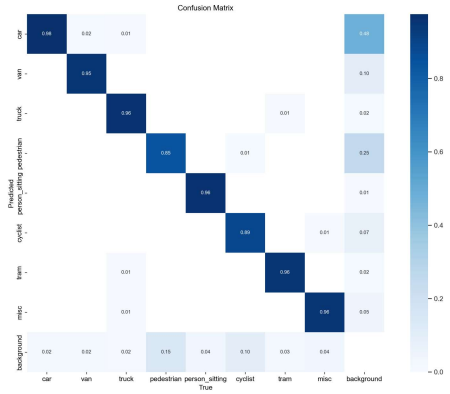
(f) AT model's performance results.

Figure 23: AT result metrics

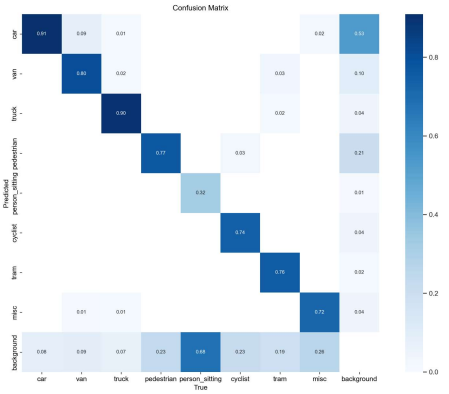
As observed from the table, all metrics show some improvement. This improvement might be attributed to the fact that the model continued training with new data, but on the same weights as $KITTI\_med\_evolve$. To determine whether the model is overfitting, its performance against adversarial attacks will be examined.

Evaluating AT against Test Set Attacks

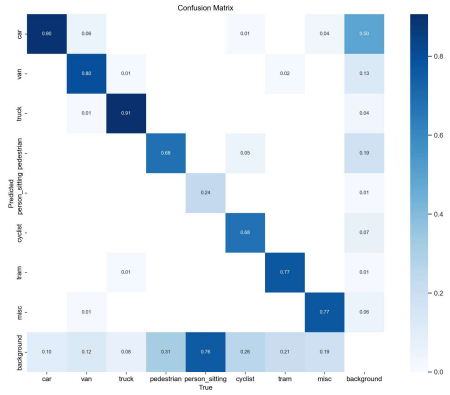
In this section, I will showcase the performance of AT when evaluating adversarial attacks against the testing dataset. The following results can be seen in *Figure 24* and *Table 16*.



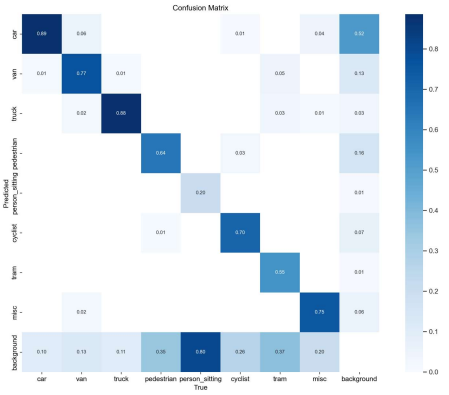
KITTI_med_evolve with AT



AP with AT



FGM with AT



PGD with AT

Figure 24: Attack confusion matrixes with AT

Metric	KITTI_med_evolve	AP	FGM	PGD
Before AT				
F1	0.90 at 0.571	0.84 at 0.421	0.21 at 0.159	0.21 at 0.523
Precision	1.00 at 0.956	1.00 at 0.960	1.00 at 0.977	1.00 at 0.986
Recall	0.95 at 0.000	0.92 at 0.000	0.32 at 0.000	0.60 at 0.000
Precision-Recall mAP@0.5	0.918	0.869	0.172	0.171
Average Effectiveness (in respect to the model)	-	**-9.1%**	**-62.8%**	**-54.5%**
After AT				
F1	0.94 at 0.595	0.79 at 0.576	0.76 at 0.502	0.73 at 0.658
Precision	1.00 at 0.963	1.00 at 0.987	1.00 at 0.992	1.00 at 0.988
Recall	0.98 at 0.000	0.85 at 0.000	0.83 at 0.000	0.81 at 0.000
Precision-Recall mAP@0.5	0.962	0.792	0.768	0.745
Average Effectiveness (in respect to the model and attacks)	**+3.10%**	**-5.6%**	**+191.95%**	**+154.57%**

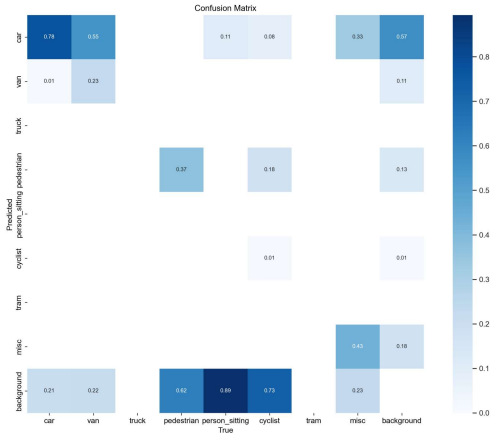
Table 16: Adversarial Training effectiveness against adversarial attacks on testing dataset.

Average effectiveness Let $M_{\text{model}}[i]$ denote the default metric values of the model before the defence is applied, $M_{\text{def}}[i]$ denotes the metric values after the defence is applied, and $M_{\text{att}}[i]$ denote the metric values under attack, for $i \in \{1, 2, 3, 4\}$, respectively. Before applying the defence, the effectiveness of the attack is calculated with respect to the default model metrics as $E_{att}[i] = \frac{M_{\text{model}}[i] - M_{\text{att}}[i]}{M_{\text{att}}[i]}$. After applying the defence, the effectiveness for each metric is calculated with respect to the model metrics under the attacks as $E[i] = \frac{M_{\text{def}}[i] - M_{\text{att}}[i]}{M_{\text{att}}[i]}$ and the effectiveness of the defence with respect to the model as $E_{model\_with\_def}[i] = \frac{M_{\text{model}}[i] - M_{\text{def}}[i]}{M_{\text{def}}[i]}$. Lastly, for all cases the average effectiveness, E_{avg}, is computed as $E_{\text{avg}} = \frac{1}{4}\sum_{i=1}^{4} E[i]$.

Upon examining the results, it is evident that AT has dramatically increased the model's robustness. The confusion matrix reveals that the model is far less likely to misclassify images. The metrics after the defence is implemented are similar to a model that has not been attacked. All metrics for FGM and PGD attacks show significant improvement, except for *Precision*, for reasons discussed earlier. In contrast, the AP attack is the only one that shows reduced robustness after AT implementation. This reduction in performance is likely since the AP attack did not fool the model initially, making the training for it redundant. In summary, the AT method demonstrates excellent results in enhancing the model's robustness in the testing environment.

Evaluating AT on Real-life video attacks

The subsequent results showcase the effectiveness of the AT defence, as presented in *Table 17* and *Figure 25*.



(a) FGM attack

(b) PGD attack

(c) AP attack

Figure 25: Attack confusion matrixes with AT in real-life environment

Metric	FGM	PGD	AP	Average Effectiveness
Before AT				
F1	0.21 at 0.130	0.17 at 0.644	0.47 at 0.132	-
Precision	1.00 at 0.957	1.00 at 0.989	1.00 at 0.983	-
Recall	0.56 at 0.000	0.64 at 0.000	0.80 at 0.000	-
Precision-Recall mAP@0.5	0.175	0.128	0.483	-
After AT				
F1	0.42 at 0.108	0.40 at 0.253	0.38 at 0.054	**+100% | +135% | -19%**
Precision	1.00 at 0.991	1.00 at 0.996	1.00 at 0.989	**+3.6% | +0.7% | +0.6%**
Recall	0.63 at 0.000	0.61 at 0.000	0.51 at 0.000	**+12.5% | -4.7% | -36.3%**
Precision-Recall mAP@0.5	0.412	0.393	0.368	**+135% | +207% | -24%**

Table 17: Adversarial Training effectiveness against adversarial attacks on real-time video.

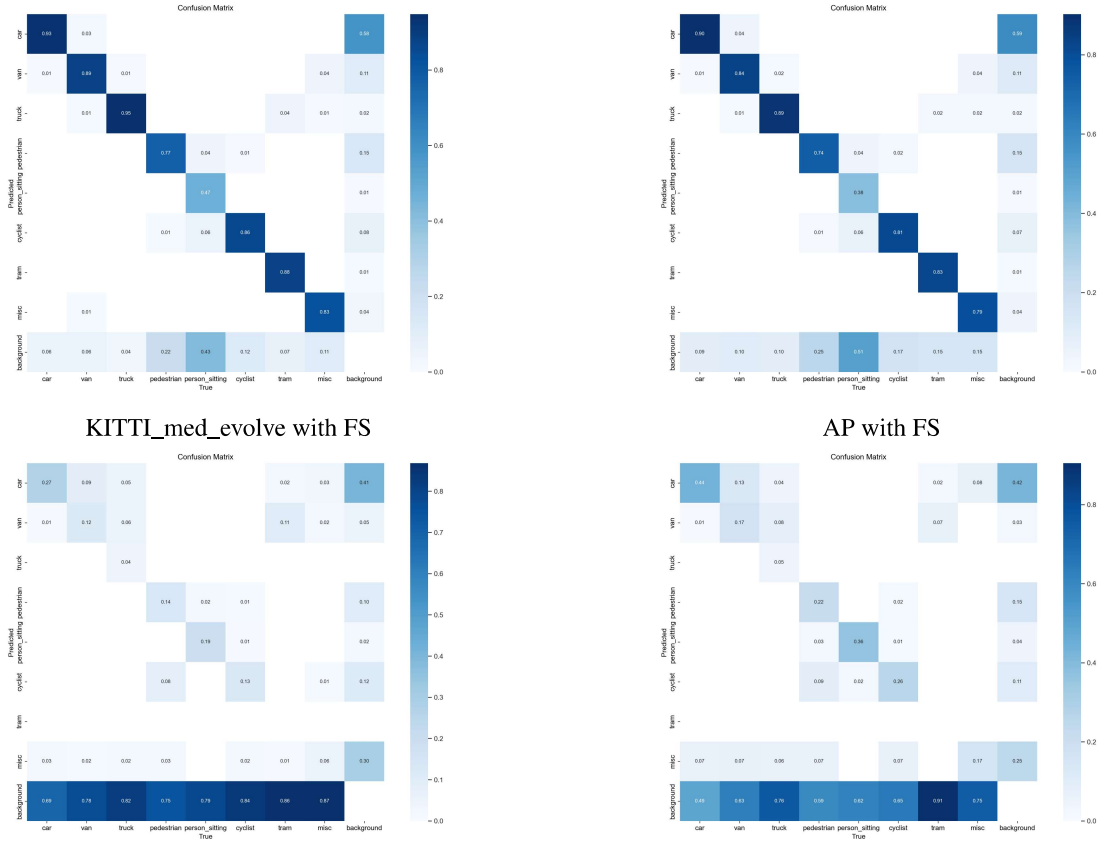
Average Effectiveness for real-time video Because there is no default model to compare the results, the average effectiveness is calculated for each metric as $E_{metric} = \frac{M_{\text{def}} - M_{\text{att}}}{M_{\text{att}}}$, M_{def} being the metric score after the defence is implemented, and M_{att} - metric score for the attack without the defence implemented.

Upon examining the results from real-life scenarios, we can see that the robustness of the model has improved in nearly all metrics, similar to the testing results. While the confusion matrixes show some misclassification is done, it is less sparse, and the objects are not as prone to be misclassified as background objects. The only exception is

the AP attack. For FGM and PGD, *Precision* and *Recall* show little or no improvement, which could be attributed to the nature of the dataset, as discussed in previous sections. However, the *F1 score* and *Precision-Recall scores* have increased in the range of *100-207%*, indicating that the implementation of AT has made the model more robust. The only outlier, as in the attack evaluations, is the AP attack, showing decreased performance in all metrics except for *Precision*, which exhibits minimal improvement. This is likely due to the same reasons mentioned in the previous section – since the model was not fooled by the attack, the training became detrimental. Despite this, the AT defence mechanism demonstrates excellent performance in improving the model's robustness against gradient-based attacks.

5.4 FS results

Evaluating FS against Test Set Attacks
In this section, we will discuss the results of evaluating the FS defence method on the testing dataset, as presented in *Table 18* and *Figure 26*. We have already examined the results of FS in the *Implementation* section when evaluating the method on the validation dataset.



KITTI_med_evolve with FS

AP with FS

FGM with FS

PGD with FS

Figure 26: Attack confusion matrixes with FS

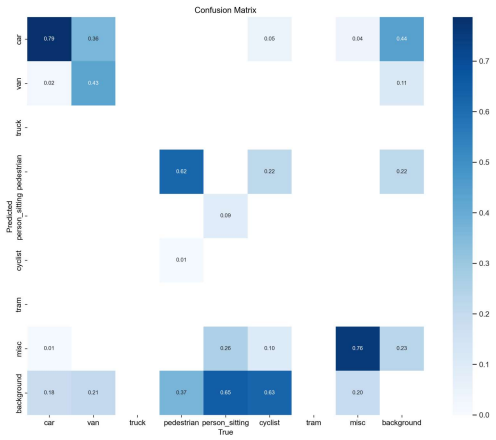
Metric	KITTI_med_evolve	AP	FGM	PGD
Before FS				
F1	0.90 at 0.571	0.84 at 0.421	0.21 at 0.159	0.21 at 0.523
Precision	1.00 at 0.956	1.00 at 0.960	1.00 at 0.977	1.00 at 0.986
Recall	0.95 at 0.000	0.92 at 0.000	0.32 at 0.000	0.60 at 0.000
Precision-Recall mAP@0.5	0.918	0.869	0.172	0.171
Average Effectiveness (in respect to the model)	**-**	**-9.1%**	**-62.8%**	**-54.5%**
After FS				
F1	0.84 at 0.375	0.81 at 0.377	0.16 at 0.105	0.14 at 0.380
Precision	1.00 at 0.960	1.00 at 0.988	1.00 at 0.960	1.00 at 0.962
Recall	0.94 at 0.000	0.90 at 0.000	0.27 at 0.000	0.45 at 0.000
Precision-Recall mAP@0.5	0.882	0.836	0.131	0.112
Average Effectiveness (in respect to the model and attacks)	**-2.91%**	**-2.39%**	**-15.82%**	**-23.21%**

Table 18: Feature Squeezing effectiveness against adversarial attacks on testing dataset.

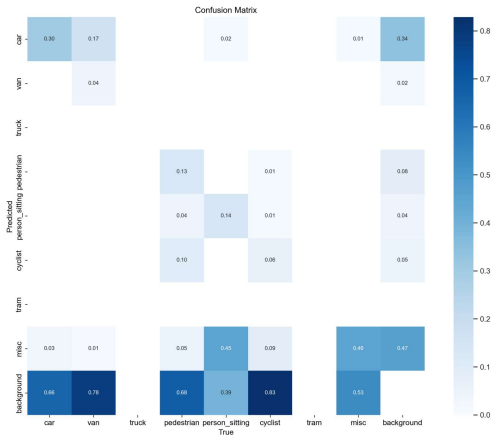
Upon analyzing the results, we observe that the defence failed to improve the model's robustness. The confusion matrix reveals that the disparity between misclassifications remained similar to the attack matrices even after implementing FS. Furthermore, while the default model metrics display good performance with only a *2.91%* decrease, the FS performance on the attack methods exhibited poor results. The metric scores after the defence implementation decreased by *2.39%* AP and *15.82%* and *23.21%* for FGM and PGD, respectively. This drastic reduction in accuracy can be attributed to the implementation of FS. As the method reduces the bit-depth of the image, it can make adversarial perturbations more visible and increase the likelihood of fooling the model due to the loss of information and features. Nonetheless, these findings offer important information on FS as a preprocessing defence mechanism.

Evaluating FS against Real-life Attacks

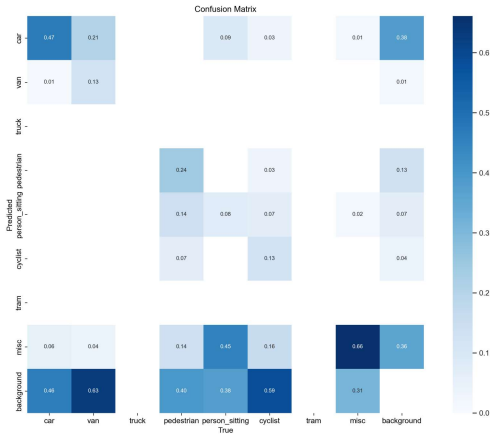
The results of real-life video testing for FS can be seen in *Table 19* and the plots in *Figure 27*.



(a) FGM attack



(b) PGD attack



(c) AP attack

Figure 27: Attack confusion matrixes with FS in real-life environment

Metric	FGM	PGD	AP	Average Effectiveness
Before FS				
F1	0.21 at 0.130	0.17 at 0.644	0.47 at 0.132	-
Precision	1.00 at 0.957	1.00 at 0.989	1.00 at 0.983	-
Recall	0.56 at 0.000	0.64 at 0.000	0.80 at 0.000	-
Precision-Recall mAP@0.5	0.175	0.128	0.483	-
After FS				
F1	0.17 at 0.084	0.14 at 0.372	0.43 at 0.253	**-19% | -18% | +6.9%**
Precision	1.00 at 0.972	1.00 at 0.994	1.00 at 0.978	**+1.6% | +0.5% | -0.5%**
Recall	0.51 at 0.000	0.60 at 0.000	0.76 at 0.000	**-8.9% | -6.3% | -5%**
Precision-Recall mAP@0.5	0.127	0.104	0.459	**-27% | -19% | -5%**

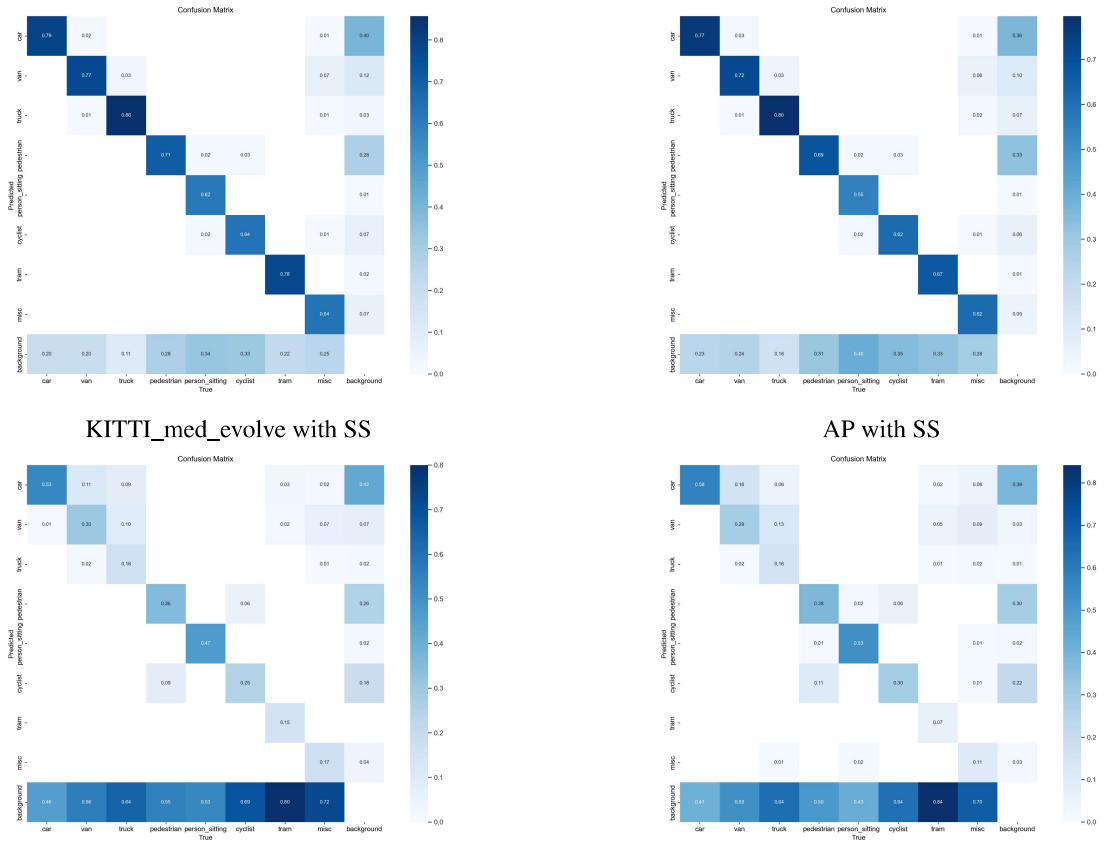
Table 19: Feature Squeezing effectiveness against adversarial attacks on real-time video.

As seen from the results, similarly to the testing environment, the FS defence did not make the model robust in real-time video, further decreasing the accuracy. The confusion matrices show an increased likelihood of misclassifying data as either background or miscellaneous objects. Almost all metrics, except for the slight improvement in the *Precision metric* for FGM and PGD (*1.6% and 0.5%*), decreased by a range of *[0.5 – 27]%* after implementing the defence. The drop in effectiveness is not only due to the method's characteristics but also the nature of the gathered data. By

examining both *Tables 18* and *19*, and *Figures 26* and *27*, the FS method did not make the model more robust - in most scenarios, it made it less accurate compared to the metrics to the attack methods.

5.5 SS results

Evaluating SS against Test Set Attacks
The last defence method SS results on the testing dataset can be seen in *Table 20* and *Figure 28*.



KITTI_med_evolve with SS

AP with SS

FGM with SS

PGD with SS

Figure 28: Attack confusion matrixes with SS

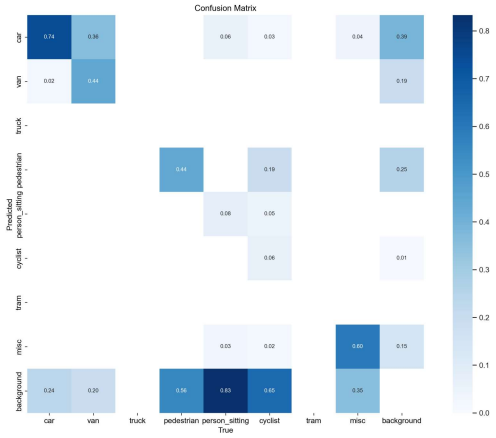
Metric	KITTI_med_evolve	DPT	FGM	PGD
Before SS				
F1	0.90 at 0.571	0.84 at 0.421	0.21 at 0.159	0.21 at 0.523
Precision	1.00 at 0.956	1.00 at 0.960	1.00 at 0.977	1.00 at 0.986
Recall	0.95 at 0.000	0.92 at 0.000	0.32 at 0.000	0.60 at 0.000
Precision-Recall mAP@0.5	0.918	0.869	0.172	0.171
Average Effectiveness (in respect to the model)	-	**-9.1%**	**-62.8%**	**-54.5%**
After SS				
F1	0.80 at 0.353	0.75 at 0.402	0.39 at 0.268	0.30 at 0.342
Precision	1.00 at 0.957	1.00 at 0.957	1.00 at 0.966	1.00 at 0.976
Recall	0.89 at 0.000	0.84 at 0.000	0.49 at 0.000	0.51 at 0.000
Precision-Recall mAP@0.5	0.831	0.769	0.337	0.253
Average Effectiveness (in respect to the model and attacks)	**-6.73%**	**-7.73%**	**+58.68%**	**+19.04%**

Table 20: Spatial Smoothing effectiveness against adversarial attacks on testing dataset.

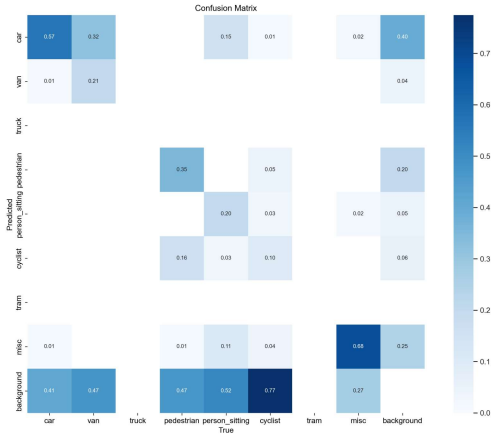
As seen from the results, the model is made more robust when it comes to FGM and PGD attacks. But the confusion matrixes show that the model misclassifies the images often as background objects. Furthermore, while the accuracy for this method is slightly lower than the FS, with a decrease of *6.73%*, the accuracy when it comes to adversarial examples is increased, boosting the performance, except for AP with *-7.73%*, *+58.68%* and *+19.04%* for FGM and PGD. Differently from FS results, the SS deals well with FGM and PGD attacks. This change is likely due to the different approaches of the methods - the SS method does not make the perturbations more visible, in fact, the input blurring makes the attacks less visible and it resembles random noise, with whom the model is more capable of dealing. Nevertheless, the SS method's low scores for metrics, indicate that it too suffers from the same issues as FS – *the transformations made during preprocessing reduces models performance and accuracy when dealing with gradient-based attacks*.

Evaluating SS against Real-life Attacks

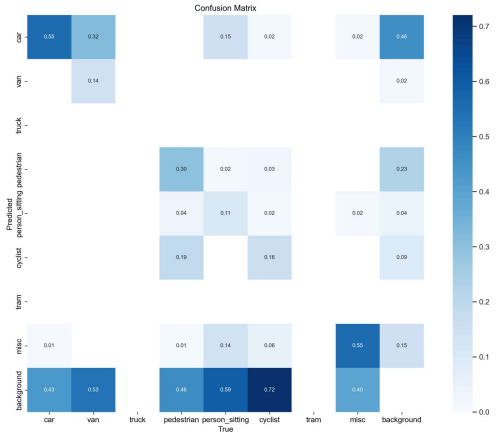
Table 21 and *Figure 29* represent the SS results when evaluating it in a real-life environment.



(a) FGM attack



(b) PGD attack



(c) AP attack

Figure 29: Attack confusion matrixes with SS in real-life environment

Metric	FGM	PGD	AP	Average Effectiveness
Before SS				
F1	0.21 at 0.130	0.17 at 0.644	0.47 at 0.132	-
Precision	1.00 at 0.957	1.00 at 0.989	1.00 at 0.983	-
Recall	0.56 at 0.000	0.64 at 0.000	0.80 at 0.000	-
Precision-Recall mAP@0.5	0.175	0.128	0.483	-
After SS				
F1	0.36 at 0.159	0.26 at 0.182	0.48 at 0.149	**+71.4%	+52.9%	+2.1%**
Precision	1.00 at 0.960	1.00 at 0.967	1.00 at 0.984	**+0.3%	-2.2%	+0.1%**
Recall	0.70 at 0.000	0.65 at 0.000	0.68 at 0.000	**+25.0%	+1.6%	-15.0%**
Precision-Recall mAP@0.5	0.331	0.215	0.463	**+89.1%	+67.2%	-4.1%**

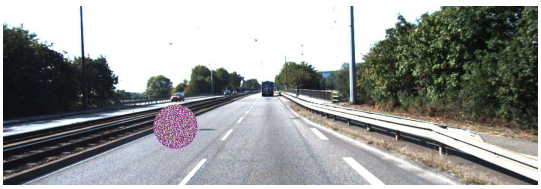
Table 21: Spatial Smoothing effectiveness against adversarial attacks on real-time video.

In the real-life environment, the defence demonstrates a considerable improvement in model robustness. Although the confusion matrices indicate some reduction in misclassification, the metric results are more prominent. With the exception of the AP attack due to previously discussed reasons, there is a significant increase in FGM and PGD *F1* and *Precision-Recall scores* within the ranges of *[52.9-71.4]%* and *[67.2-89.1]%* respectively. These findings highlight that *SS substantially enhances the metrics in real-life and testing scenarios when dealing with attacks, but has reduced accuracy*. The possible explanation for this outcome is the difference in the nature of the data; the reduction in image sharpness when processing a video might render the SS method more suitable. In conclusion, the outcomes derived from the results indicate that SS demonstrates a level of viability as an independent method of defense, but there is a considerable trade-off between robustness and models performance.

5.6 Defence visualization

In this section, *Figures 30* and *31* illustrate the difference between image preprocessing by FS and SS methods.

Default image with FS



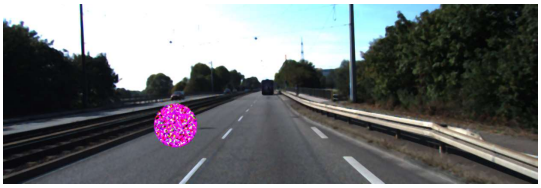
AP image with FS

FGM image with FS

PGD image with FS

Figure 30: Images generated by FS

Default image with SS



AP image with SS

FGM image with SS

PGD image with SS

Figure 31: Images generated by SS

As observed in the figure, both defence techniques alter the key characteristics of the targeted image. FS achieves this by reducing the *bit-depth*, resulting in a brighter image with higher contrast compared to the original, making the perturbations more visible. On the other hand, SS applies a "*blur*" effect, which makes certain aspects of the image less visible, slightly reducing the perturbations. In essence, both methods try to accomplish a similar objective - modifying the image in a manner that reduces its susceptibility to adversarial attacks.

5.7 Summary

Upon reviewing the results from each defence, it is evident that AT stands out as the most effective method in enhancing the robustness of the model in both testing and real-life scenarios. This is demonstrated by **AT outperforming SS and FS in every metric, except for the AP attack**, which consistently emerged as an outlier in all evaluations for the defences. While FS decreased the model's accuracy in testing and real-life scenarios, SS produced a different outcome in those environments, surpassing FS across all metrics, except when it came to the AP attack. In conclusion, although the defences exhibited varying results in terms of the model's robustness, this evaluation provides valuable insights into the unique weaknesses and strengths of each method when implemented individually.

6 Conclusion

In this research, I aimed to create visible and effective adversarial attacks and defence methods to enhance the robustness of the YOLOv5n custom model when handling malicious inputs. A robust model, $KITTI\_med$, was trained on the autonomous vehicle dataset KITTI, which was further optimized to create the $KITTI\_med\_evolve$ model. This resulted in an effective object detection model that optimally generalized the data. All the attacks, AP, PGD, and FGM, generated distinct adversarial images due to their different architectures. The most effective, in terms of metrics, and visible attack was FGM, while PGD came close in terms of reducing the model's accuracy. The AP method stood out as an outlier, as it failed to effectively fool the model as intended and demonstrated no improvement with varying parameters and testing environments. However, these attacks demonstrated their unique strengths and characteristics. My evaluation revealed that PGD was the most effective attack, considering its more subtle perturbations, decreased metrics of the $KITTI\_med\_evolve$ model and that the defences did not improve the model's robustness as much as with FGM when the defensive methods were applied. This research highlights that attacks creating gradient perturbations in the entire image are more effective than those generating localized small patches of perturbations, especially considering computational complexity and hardware limitations. In contrast to other studies in the field, this research visually displays the perturbations, providing a more tangible illustration of the distortions. The defence methods also exhibited varying results. All defences failed to improve the model when dealing with AP attacks, because of its abnormal behaviour, but because the attack was not effective the model's accuracy was reduced only slightly. AT was the most effective method by a significant margin, outperforming FS and SS in all metrics and enabling the model to handle adversarial images. Surprisingly, FS proved ineffective in enhancing the model's robustness, resulting in decreased accuracy and worse performance compared to the metrics before the application. In contrast, the performance of SS significantly outperformed that of FS in test environments. More importantly, it demonstrated robust effectiveness in real-world applications. While FS and SS had different outcomes in terms of model robustness, they

demonstrated differences in input images after preprocessing. AT emerged as the most straightforward and effective method for improving model robustness, while FS and SS exhibited decreased performance when implemented alone. This suggests that AT can serve as a standalone defence method, while FS and SS should be used in conjunction with other defence methods to bolster their performance instead of relying solely on them for model protection. In conclusion, the developed model effectively detected objects, and the attacks offered valuable insight into the risks and effectiveness of each method within the given environment. The implemented defences showcased their limitations and strengths when applied individually in terms of model robustness. This research presents new findings and underscores the importance of effective defence techniques while highlighting the characteristics, effectiveness, and discovered limitations of the adversarial attacks that malicious actors may employ. When discussing the broader impact of this research, it contributes to the understanding of AI security in autonomous vehicles. It emphasizes the need to develop robust models that can withstand adversarial attacks and proposes effective defence mechanisms to counter such threats.

6.1 Future Directions

While this research paper effectively examines various attack and defence methods in different environments, there is potential for further exploration and expansion in future work.

Model and Dataset Improvements:
Future research could explore implementing larger YOLOv5 models, such as YOLOv5s or YOLOv5m, to evaluate their impact on the model's robustness. Additionally, increasing the dataset size or incorporating more classes and diverse images may lead to better generalization and performance.

Adversarial Attack Enhancements:
The current adversarial attack methods could be refined to be less visible, making them more challenging to detect. Furthermore, exploring additional white-box attacks and introducing new black-box attacks could provide a more comprehensive understanding of potential threats. Revisiting the AP implementation and improving it to become an effective adversarial attack method could be a valuable area of investigation. Furthermore, exploring new attack methods, such as poisoning or extraction-based attacks, alongside the existing ones will provide a more comprehensive understanding of adversarial attacks in this domain.

Defence Mechanism Improvements:
Comparing the current defence methods to a model that incorporates all defences could offer insights into their combined effectiveness. Implementing novel defence mechanisms, such as postprocessing techniques or adversarial attack detection systems, may further enhance model robustness. Additionally, improving the FS and SS methods to increase model robustness is another potential avenue for future work.

Real-Time Object Detection GUI:
To confidently assess the model's robustness in real-life scenarios, future research could develop a real-time object detection *graphical user interface (GUI)* that includes real-time acting defence and attack methods. This implementation would provide more accurate results when testing the model's performance in real-life and real-time scenarios.

References

[1] Trusted-ai/adversarial-robustness-toolbox/issues/1907. `https://github.com/Trusted-AI/adversarial-robustness-toolbox/issues/1907`, 2023. Accessed on April 3, 2023.

[2] Khaled R. Ahmed. Smart pothole detection using deep learning based on dilated convolution. *Sensors*, 21(24), 2021.

[3] Ebubekir BUBER and Banu DIRI. Performance analysis and cpu vs gpu comparison for deep learning. In *2018 6th International Conference on Control Engineering & Information Technology (CEIT)*, pages 1–6, 2018.

[4] Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, 2014.

[5] Yao Deng, Xi Zheng, Tianyi Zhang, Chen Chen, Guannan Lou, and Miryung Kim. An analysis of adversarial attacks and defenses on autonomous driving models, 2020.

[6] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, and D. Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1625–1634, 2018.

[7] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[8] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

[9] R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Pearson, 3rd edition, 2008.

[10] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.

[11] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[12] Wei Jia, Zhaojun Lu, Haichun Zhang, Zhenglin Liu, Jie Wang, and Gang Qu. Fooling the eyes of autonomous vehicles: Robust physical adversarial examples against traffic sign recognition systems, 2022.

[13] G. Jocher. Yolov5 by ultralytics, 2020.

[14] L. Jones. Uber's self-driving car kills pedestrian in first fatal autonomous crash, 2018.

[15] JUtah. London 4k - monday morning - driving downtown uk. `https://www.youtube.com/watch?v=y1OCipyZefA`, 2018. Accessed: 2023-04-10.

[16] Parvinder Kaur, Baljit Singh Khehra, and Er. Bhupinder Singh Mavi. Data augmentation for object detection: A review. In *2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 537–543, 2021.

[17] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale, 2017.

[18] Mark Lee and Zico Kolter. On physical adversarial patches for object detection, 2019.

[19] Li Li, Miloš Doroslovački, and Murray H. Loew. Approximating the gradient of cross-entropy loss function. *IEEE Access*, 8:111626–111635, 2020.

[20] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[21] Xin Liu, Huanrui Yang, Ziwei Liu, Linghao Song, Hai Li, and Yiran Chen. Dpatch: An adversarial patch attack on object detectors, 2019.

[22] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

[23] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[24] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[25] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[26] Trusted-AI/adversarial robustness toolbox. Issue #2086: Add support for pytorch lightning. `https://github.com/Trusted-AI/adversarial-robustness-toolbox/issues/2086`, 2023. Accessed on March 24th, 2023.

[27] Yash Sharma and Pin-Yu Chen. Bypassing feature squeezing by increasing adversary strength. 2018.

[28] Daria Snegireva and Anastasiia Perkova. Traffic sign recognition application using yolov5 architecture. In *2021 International Russian Automation Conference (RusAutoCon)*, pages 1002–1007, 2021.

[29] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses, 2020.

[30] Trusted-AI. Adversarial robustness toolbox. `https://github.com/Trusted-AI/adversarial-robustness-toolbox`, 2023.

[31] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy, 2019.

[32] Ultralytics. Yolov5 hyperparameter evolution. `https://docs.ultralytics.com/yolov5/tutorials/hyperparameter_evolution/`, 2023. Accessed on April 28th, 2023.

[33] C.-Y. Wang and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[34] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *Proceedings 2018 Network and Distributed System Security Symposium*. Internet Society, 2018.

[35] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning, 2018.

[36] Fangbo Zhou, Huailin Zhao, and Zhen Nie. Safety helmet detection based on yolov5. In *2021 IEEE International Conference on Power Electronics, Computer Applications (ICPECA)*, pages 6–11, 2021.

[37] Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven Hoi, and Weinan E. Towards theoretically understanding why sgd generalizes better than adam in deep learning, 2021.