ISA Transactions 132 (2023) 69–79

Contents lists available at ScienceDirect

ISA Transactions

journal homepage: www.elsevier.com/locate/isatrans

Towards robust autonomous driving systems through adversarial test set generation

Devrim Unal [a,*], Ferhat Ozgur Catak [b], Mohammad Talal Houkan [c], Mohammed Mudassir [c], Mohammad Hammoudeh [d]

[a] KINDI Center For Computing Research, Qatar University, Doha 2713, Qatar
[b] Electrical Engineering and Computer Science Department, University of Stavanger, Rogaland 4021, Norway
[c] Department of Mechanical and Industrial Engineering, Qatar University, Doha 2713, Qatar
[d] Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia

ARTICLE INFO

Article history:
Received 5 April 2022
Received in revised form 9 November 2022
Accepted 9 November 2022
Available online 17 November 2022

Keywords:
Risk-aware autonomous systems
DL
Test set generation
Uncertainty

ABSTRACT

Correct environmental perception of objects on the road is vital for the safety of autonomous driving. Making appropriate decisions by the autonomous driving algorithm could be hindered by data perturbations and more recently, by adversarial attacks. We propose an adversarial test input generation approach based on uncertainty to make the machine learning (ML) model more robust against data perturbations and adversarial attacks. Adversarial attacks and uncertain inputs can affect the ML model's performance, which can have severe consequences such as the misclassification of objects on the road by autonomous vehicles, leading to incorrect decision-making. We show that we can obtain more robust ML models for autonomous driving by making a dataset that includes highly-uncertain adversarial test inputs during the re-training phase. We demonstrate an improvement in the accuracy of the robust model by more than 12%, with a notable drop in the uncertainty of the decisions returned by the model. We believe our approach will assist in further developing risk-aware autonomous systems.

1. Introduction

Over the past decade, Deep Learning (DL) succeeded in a wide range of applications, partly due to the use of powerful learning algorithms that can learn complex relationships from large-scale datasets. More recently, DL schemes were applied successfully in various tasks related to autonomous driving, such as perception, prediction, planning, decision-making, and control. Developing robust DL schemes continues to attract more attention to safety–critical application domains. Despite the success of DL [1], there are still challenges in applying DL to many real-world applications. One of the biggest obstacles to using DL in many applications is the lack of explainability of how and why DL networks work.

In some cases, e.g., in medical applications, the neural network function must be explained to the user [2]. Another critical issue is that Deep Neural Networks (DNNs) are very sensitive to the change in the distribution of inputs. For example, the image classification network trained on the ImageNet dataset [3] cannot be directly used to classify images of different domains, such as medical images. If we want to use a model trained on a large dataset, e.g., ImageNet [4], to classify images in other domains, the model will not work well. The main challenge here, is to train a model that performs on all different data types and in real-life circumstances with high uncertainty and noise.

When deploying a model to a new environment, it is critical to test the model to ensure accurate results. However, there are many ways to test a model. Traditional testing methods, including statistical testing, modeling, and validation, do not produce a robust enough test set that can cover all possible scenarios. Instead, to build DL models ready for industrial applications, we need to follow software test engineering methodologies [5] to develop and validate robust ML models. There are many different testing methodologies in software testing, such as white-box testing, black-box testing, and grey-box testing [6]. The grey-box testing is a methodology that combines white-box testing and black-box testing. Grey-box testing involves both the internal structure of the system and the environment in which the system operates. The system's internal structure is used to generate test data, and the environment is used to validate the test. Grey-box testing is a robust testing methodology that can test the system from different perspectives.

* Corresponding author.
E-mail addresses: dunal@qu.edu.qa (D. Unal), f.ozgur.catak@uis.no (F.O. Catak), mh1208170@qu.edu.qa (M.T. Houkan), mh1204033@qu.edu.qa (M. Mudassir), M.Hammoudeh@kfupm.edu.sa (M. Hammoudeh).

https://doi.org/10.1016/j.isatra.2022.11.007
0019-0578/

In this paper, we propose a grey-box testing methodology that can test the system from the internal structure of the system, the environment, and the user. The proposed method uses a system model to generate test data, a test oracle to validate the test, i.e., prediction performance, and a user model to evaluate the system, i.e., DNN model. There are different approaches to developing robust DL schemes, such as modifying the data, the model itself, adding auxiliary models, and adversarial test input generation. In this work, our focus is the test input generation-based robust DL scheme development in autonomous driving systems. Two significant gaps in recent works are the interpretability of DL models and training robust models in the presence of different training and test distributions [7]. This paper addresses both of these gaps by introducing a well-defined adversarial test set generation that reduces the uncertainty and produces more robust models that perform better in the presence of uncertain inputs.

We consider the two main types of uncertainty: epistemic (model) and aleatoric (data) uncertainty. When autonomous driving is considered, DL models are generally multi-output regression models. Classical means of dealing with uncertainty generally do not apply to DL. There is a need for novel methods for measuring uncertainty in multi-output regression models. According to Wickramasinghe's recent work [8], many DNN solutions for Cyber–Physical System (CPS) data only create outputs for the input instances. They do not measure uncertainty in the prediction phase, which could ultimately result in a lack of trust in these approaches and could lead to unexpected and dangerous CPS behaviors. DNN models trained on data from CPS will require an uncertainty quantification framework to quantify the model uncertainty correctly. Next, the researchers should develop procedures to deal with this uncertainty.

One of the open problems for training robust models against uncertainty is generating relevant testing data to optimize the models. For this purpose, we use existing autonomous driving data sets, including sensor data, e.g., LIDAR and GPS, and video recording data. We generate highly uncertain test inputs from these data sets to train robust models against perturbations and adversarial attacks. The generated test data will help reduce the uncertainty of the decision-making process of an autonomous driving system. To evaluate the robustness of the produced models, we use the generated test data to test the trained models. We also compare the results with the models which are not trained with the generated test data. Our recent work [9] presented a new metric for uncertainty quantification for object detection DNN models, where the experiments were conducted on the NEXET, Berkeley DeepDrive, KITTI, and Stanford datasets, together with SSD300, SSD512, and YoLo DNN models, to quantify prediction-time uncertainty. Another study [10] proposed a robust model training method (NIRVANA). NIRVANA seeks to compare the accuracy of DNN model predictions with another model and to enhance DNN model predictions using results of generated uncertainty quantification.

Connected and Autonomous Vehicles (CAVs) are subjected to cyber-attacks, particularly, zero-day attacks [11]. The limited computational power of Engine Control Units (ECU) in cars prohibits them from processing robust security protocols. Furthermore, cyber-attacks put the privacy and security of passenger data at risk. CAVs differ from traditional software in several ways including their cyber–physical nature, which comprises hardware, software, and physics. The complex operating environment of CAVs includes people, and communication platforms [12]. The development and operation of CAVs are also fraught with uncertainty due to the inherent uncertainty of the DL models, the naturally unpredictable nature of the environment, the unpredictable nature of human behavior, and the unreliable network communications among CAV parts [13,14]. The building and use of CAVs are challenging because of such uncertainty sources. Therefore, the data generated during the operation of CAVs can be used with DL models to understand CAV behaviors' uncertainties better, improve future generations of CAVs, and build novel test cases. We focus on the CAV data generated during their operation to achieve this.

There are many studies in the literature on detecting and preventing poisoning and trojan attacks; however, these studies mainly require using powerful computing resources and the cloud. Our study is different from others because we focus on test data set generation for robust ML/AI models for autonomous driving, including the development of low-energy, low-latency, and low-complexity methods that can easily integrate into the training and inference algorithms without affecting their performance.

The main contributions of this article are as follows:

1. We propose a method for generating highly uncertain test input data for a given DL model for autonomous driving systems. The generated test data are used to verify the robustness of DL models.
2. We adopted an adversarial training mitigation method for the highly uncertain test inputs to increase the robustness of the DL models.
3. We compare the results of our proposed method with the state-of-the-art testing method and show that our approach achieves better performance.

The rest of the paper is organized as follows. In Section 2, we present the related work. Section 3 presents the system model including the uncertainty-based test set generation and the adversarial re-training algorithm. Section 4 presents the experiments on generating test inputs based on uncertainty maximization. In Section 5, we present the results and discuss the outcomes. We conclude with Section 6, where we also give ideas on future research tracks.

2. Related work

DL recently gained popularity in autonomous driving and intelligent transportation system-related problems, with the ability to conduct feature selection during the learning process [15]. Some popular DL algorithms include Convolutional Neural Network (CNN), Feedforward Deep Network (FDN), Long–Short Term Memory (LSTM), Recurrent Neural Network (RNN), and Generative Adversarial Network (GAN) [15]. Boukerche et al. [16] discuss the application of DL to vision-based autonomous vehicle recognition. Various use cases include vehicle detection, vehicle make and model recognition, and vehicle re-identification. Khan et al. present [17] an LSTM-based intrusion detection system for the Internet of vehicles. Jebamikyous and Kashef [18] discuss the two prominent use cases of DL for perception in autonomous vehicles; semantic segmentation and object detection. These tasks require precise data for high-performance classification results. However, uncertainty is a factor that reduces the classification performance of DL algorithms.

Various research works addressed the uncertainty in autonomous systems and proposed multiple methods to increase the performance of ML/DL algorithms in the presence of uncertainty. In [19], the authors analyze various sources of uncertainty in autonomous systems, such as execution uncertainty, timing uncertainty, adversarial attacks, and uncertainty of the behavior of Neural Networks (NNs). The authors propose a cross-layer weakly-hard design framework to tackle the execution uncertainty problem and safety verification of NNs through reachability analysis. Another study [20] addresses safe and adaptive autonomous navigation in uncertainty. The common shortcoming

of the approaches mentioned above is that they require knowledge of the NN architecture to improve performance. In our proposed work, test-set generation based on uncertain inputs and subsequent re-training is utilized for this purpose, and previous knowledge of the NN architecture is not needed.

Uncertainty quantification is a method to rank the generated inputs based on their corresponding uncertainty metrics values (variance, maximum probability), re-training the model with the highly uncertain new generated inputs. In [21], the authors apply the Monte-Carlo (MC) dropout method to measure uncertainty in inputs for autonomous driving. The deep ensemble is a non-Bayesian uncertainty quantification method [22]. Adversarial ML methods are based on generating challenging and misleading inputs using attacks and then adding generated data or images to the original test set. In [23], Ma et al. use various adversarial training methods, such as the DeepFool, Basic Iterative Method, Carlini–Wagner (CW), Fast Gradient Sign Method (FGSM), and the Jacobian-Based Saliency Map Attack. The main aim of this study is to increase the uncertainty in the re-training phase to improve the model prediction performance.

In DeepXplore, new metrics such as Network Coverage (NC) and the number of activated neurons in the prediction have been used [24]. DeepTest generates new inputs for autonomous driving DL models where different image transformations lead to different NC values [25]. DeepMutation uses mutation score, a parameter used in traditional software engineering for testing DNN models. Their proposed method creates multiple new mutant DNN models from a given model using additional methods like neuron switch or layer removal [26]. DeepCT is a test coverage metric indicating that within a given DNN layer, all tuples of neurons in that DNN layer must be covered by at least one test image [27]. DeepHunter is a fuzzing-based test generation algorithm to hunt defects in DL models. The coverage metrics guide the fuzzing [28]. Ma et al. [29] define uncertainty-based metrics for test case generation in DL systems. The authors argue that the most critical test inputs are those with high uncertainty, a principle we adopt in this paper for generating more realistic autonomous driving test sets.

Djenouri et al. proposed an accurate object detection model by adopting the Granular Region Convolution Neural Network (GR-CNN) to process vehicle image data to reduce accidents through a smart road system [30]. Mekala et al. highlighted the importance and feasibility of DL techniques for autonomous vehicles to accomplish an intelligent driving system without human interaction [31]. They have reviewed the Lidar-based DL strategies to address the research challenges in autonomous driving through a comprehensive analysis of Semantic Segmentation, Data Representation, Feature Extraction, Dynamic Object Detection, Data Fusion, and Autonomous Driving-Multi-Objective tracking mechanisms.

The existing literature addresses uncertainty quantification and various methods to train better-performing ML/AI models in the presence of uncertainty. The current literature lacks the issue of training robust models against uncertainty by generating relevant testing data to optimize the models. There is no systematic approach to test data generation based on adjustable parameters that can be measured which represents the uncertainty of the test data, and evaluates the effect of changing these parameters on the performance.

3. Uncertainty types and quantification in DL

To mitigate the overconfidence problem in DL models, we need to train the model to be uncertain about its predictions. This section introduces methods from the literature used to measure the output uncertainty of DL models. Table 1 shows the two main types of methods for measuring the output uncertainty of DL models. Bayesian methods are based on Bayesian inference, whereas non-Bayesian methods are based on other methods such as Monte Carlo sampling, importance sampling, and bootstrap sampling. Bayesian methods are usually more accurate but require more extensive training data and are computationally expensive compared to non-Bayesian methods. For these reasons, non-Bayesian methods are more prevalent in practice.

We present our view of the taxonomy of uncertainty types in Fig. 1. This taxonomy is motivated by the notion of *epistemic* and *aleatoric* uncertainty in the statistical literature and the recent work on DL uncertainty in [32]. The boxes in the diagram show various sources of uncertainty, while the arrows indicate how they propagate through the DL model. The middle region shows the sources of uncertainty considered in this paper.

Epistemic uncertainty, also known as model uncertainty, arises from the fact that the model has been trained on an imperfect dataset and in an imperfect environment. It corresponds to the model's inherent uncertainty, which one can reduce during training by providing more data. *Aleatoric* uncertainty, or data uncertainty, stems from imperfect and incomplete data. Whereas *Classical* uncertainty, or label uncertainty, arises from the fact that the ground truth labels are imperfect and incomplete. Classical uncertainty corresponds to the inherent uncertainty in the labels, which can be reduced by providing more data. *Prediction* uncertainty, or output uncertainty, arises from the fact that the model has been trained on an imperfect dataset and in an imperfect environment. It corresponds to the inherent uncertainty in the model, which can be reduced during training by providing more data.

Model, prediction, and data uncertainty can all be exploited to improve the quality of predictions made by the model. Epistemic and Prediction uncertainty are the *reducible* components of the total uncertainty. While Classical and Aleatoric uncertainty can be considered as the *irreducible* component of the total uncertainty.

DL models are often trained using a *maximum likelihood* ML criterion, i.e., the model parameters are learned to maximize the likelihood of the training data. However, the ML criterion is known to be the Maximum a Posteriori (MAP) estimate in the Bayesian setting [33]. Thus, the ML criterion can be viewed as a special case of the MAP criterion when the prior is uniform and can be used to learn the model parameters of a DL model. However, the MAP criterion does not account for the uncertainty in the model parameters. Instead, it is accounted for by using the MAP criterion. The MAP criterion is given by

$$\theta^* = \arg\max_{\theta} \log p(\mathcal{D} \mid \theta) + \log p(\theta) \tag{1}$$

where $\mathcal{D}$ is the training data, θ are the model parameters, $p(\mathcal{D} \mid \theta)$ is the likelihood of the training data given the model parameters, and $p(\theta)$ is the prior over the model parameters.

The uncertainty in the training data can be accounted for by using the *maximum a posteriori with data uncertainty* (MAP-DU) criterion defined as

$$\theta^* = \arg\max_{\theta} \log p(\mathcal{D} \mid \theta) + \log p(\theta \mid \mathcal{D}) \tag{2}$$

where $\mathcal{D}$ is the training data, θ are the model parameters, $p(\mathcal{D} \mid \theta)$ is the likelihood of the training data given the model parameters, and $p(\theta \mid \mathcal{D})$ is the posterior over the model parameters given the training data.

The MAP-DU criterion can be used to learn the model parameters of a DL model. However, it does not account for the uncertainty in the labels. The label uncertainty can be accounted for by using the *maximum a posteriori with label uncertainty* (MAP-LU) criterion defined as

$$\theta^* = \arg\max_{\theta} \log p(\mathcal{D} \mid \theta, \mathcal{L}) + \log p(\theta \mid \mathcal{D}, \mathcal{L}) \tag{3}$$

Table 1
An overview of methods for measuring output uncertainty.

	Bayesian methods	Non-Bayesian methods
Probabilistic	- Variational Inference (VI) - Markov Chain Monte Carlo (MCMC)	- Monte Carlo Dropout (MCD) - Bootstrap Ensemble
Non-probabilistic	- None	- Confidence Interval (CI) - Importance Weighting (IW)

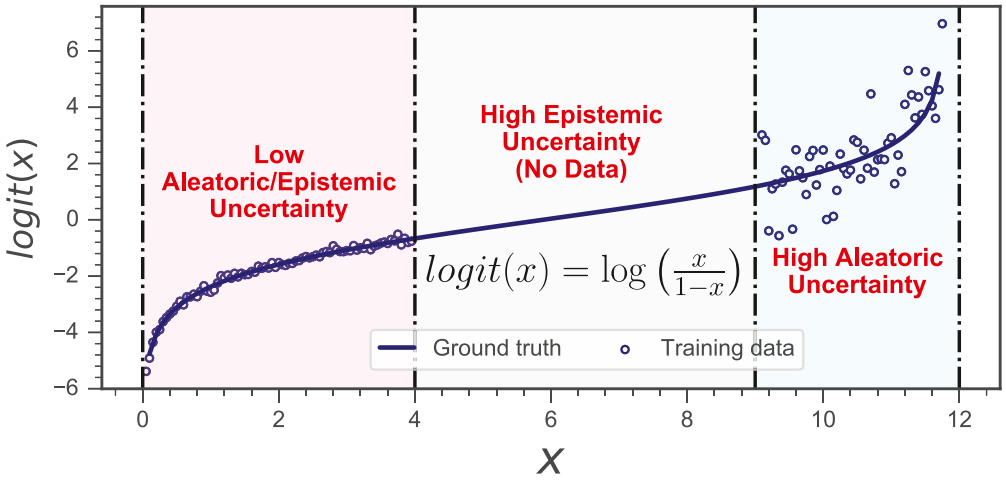

Fig. 1. Illustration of the different sources of uncertainty and how they affect the performance of a DL model.

where $\mathcal{D}$ is the training data, $\mathcal{L}$ are the labels, θ are the model parameters, $p(\mathcal{D} \mid \theta, \mathcal{L})$ is the likelihood of the training data given the model parameters and labels, and $p(\theta \mid \mathcal{D}, \mathcal{L})$ is the posterior over the model parameters given the training data and labels.

The MAP-LU criterion does not account for the uncertainty in the test data. Hence, the uncertainty in the test data can be accounted for by using the *maximum a posteriori with the test data uncertainty* (MAP-TDU) criterion given by

$$\theta^* = \arg\max_{\theta} \log p(\mathcal{D}_{\text{test}} \mid \theta, \mathcal{D}) + \log p(\theta \mid \mathcal{D}, \mathcal{D}_{\text{test}}) \tag{4}$$

where $\mathcal{D}$ is the training data, $\mathcal{D}_{\text{test}}$ is the test data, θ are the model parameters, $p(\mathcal{D}_{\text{test}} \mid \theta, \mathcal{D})$ is the likelihood of the test data given the model parameters and training data, and $p(\theta \mid \mathcal{D}, \mathcal{D}_{\text{test}})$ is the posterior over the model parameters given the training data and test data.

DL models are trained to optimize a loss function, making them susceptible to the data used to train the model. The dependence on the training data may significantly impact the model's generalization performance on unseen data during testing because DL models can extrapolate the underlying relationships in the data and generalize them to unseen data. However, a model trained on a dataset sampled from a different distribution from the testing one will not generalize well.

Fig. 1 shows the three sources of uncertainty and how they affect the performance of a DL model. Data uncertainty comes from the fact that the dataset is sampled from an underlying distribution. Irreducible uncertainty is uncertainty that one cannot remove. In contrast, model uncertainty is the uncertainty that comes from the fact that the model is not trained on the entire population. Fig. 1 shows how the three sources of uncertainty propagate through the model. The data uncertainty affects the model uncertainty, which affects the irreducible uncertainty. A model trained on a dataset generalizes better if the dataset is larger. This is because a model trained on a larger dataset is more accurate, which reduces the model's uncertainty. Data uncertainty comes from the fact that the dataset is sampled from an underlying distribution. The data uncertainty affects the model uncertainty, which in turn affects the irreducible uncertainty. The model uncertainty is the primary source of uncertainty that we focus on in this work.

A model with a high uncertainty has low performance on unseen data. In this work, we focus on the aleatoric uncertainty. There are two main ways to quantify aleatoric uncertainty, the mean–variance approach, and the predictive entropy approach. The mean–variance method quantifies the aleatoric uncertainty by the model's predictions' mean and variance. The predictive entropy approach quantifies the aleatoric uncertainty by the entropy of the model's predictions. In this article, we use the mean–variance approach.

In the mean–variance approach, the mean is the expected value of the model's predictions, and the variance is the expected value of the squared difference between the model's predictions and the mean. The mean and the variance can be estimated using the Monte Carlo method [34]. The Monte Carlo method generates samples from the model's predictions. The mean and the variance are then estimated using the generated samples.

4. System model

The main goal of this work is to build a robust DL model for Cyber–Physical Systems (CPSs), based on model re-training creating highly uncertain inputs. The proposed highly uncertain instance generation mainly consists of the following three phases: (I) is the dataset generation; (II) is the building of a dropout-based neural network at prediction time; (III) is the re-training of the model with the generated highly uncertain inputs (see Fig. 2).

4.1. Test input generation based on the non-dominated sorting genetic algorithm

NSGA-II (Non-dominated Sorting Genetic Algorithm-II) follows the general outline of a genetic algorithm with a modified mating and survival selection. In this algorithm individuals are chosen frontwise, leading to a situation where a front needs to be split because not all individuals are permitted to survive. This splitting front selects solutions based on crowding distance [35]. The main

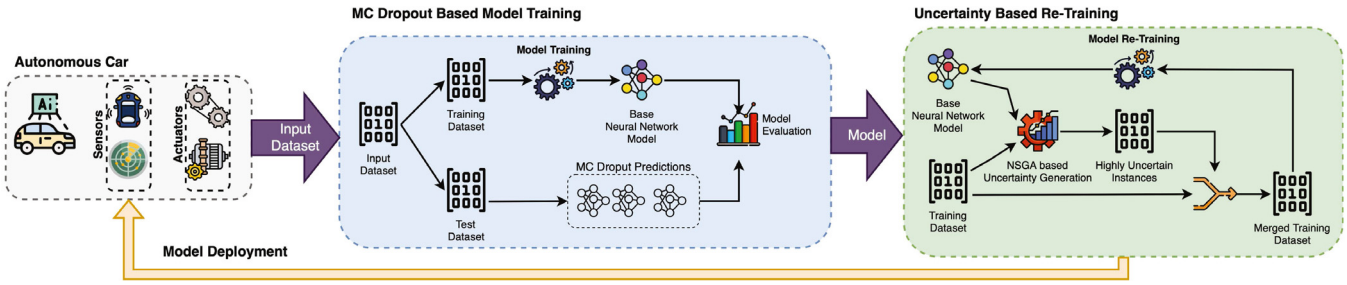

Fig. 2. System overview.

advantages of NSGA-II are that it can deal with mixed constraints, it is elitist, is very unlikely to converge prematurely to a local optimum, and is computationally efficient. Elitist algorithms, a type of evolutionary algorithm, work by making sure that the best individuals in one epoch are not discarded, and transferred directly into the next generation. The algorithm is implemented in Python through the Pymoo module. Algorithm 1 shows the steps in NSGA-II. The algorithm's inputs are; the population size P, the number of generations G, the objective function $f_i(x_i)$, the lower and upper bounds l_i, u_i of decision variables x_i. The output of NSGA-II is the set of non-dominated solutions S^*.

Algorithm 1 NSGA-II

Require: P, G, $f_i(x_i)$, l_i, u_i
Ensure: S^*
1: Initialize the population P;
2: **for** $i = 1$ to G **do**
3: Calculate the fitness of each individual x_i;
4: Sort the population P frontwise;
5: Select the non-dominated individuals S^*;
6: Calculate the crowding distance of S^*;
7: Select the best individuals with the smallest crowding distance;
8: Generate the new population P_{new};
9: $P := P_{new}$
10: **end for** RETURN S^*

Fig. 2 demonstrates the training dataset $\mathcal{D}$ is collected from CAVs in Phase I. In this study the inputs are LIDAR, and the output is a robust model that is trained to classify the objects in the LIDAR. The algorithm's primary output is the generation of a robust model that is not tricked by an adversarial test input injection. Data preparation is conducted in Phase II and DNN model training with the input dataset $\mathcal{D}$ which is carried out from the previous phase. Conventional DNN training is executed at this phase to find the optimal weights. The primary output is the base DNN model, which is the DNN model with the best prediction performance and lowest loss value over the complete training data. The most crucial contrast between the base DNN model and a conventional DNN is the prediction time activated dropouts. In contrast, a standard DNN model has no dropouts in the prediction time.

In Phase III, the model is re-trained using highly uncertain training instances to improve the DNN model's prediction performance. In this our approach, we use the training dataset and model with the NSGA-II optimization algorithm to generate new and highly uncertain instances. Then, we merge the training dataset with newly generated instances. Finally, we re-train the DNN model with the new training dataset to improve the prediction performance.

4.2. Adversarial training

In this research, we use the adversarial training method to improve the robustness of the DL model. The adversarial training mitigation method re-trains a DNN model to be robust against a specific type of perturbation. In this work, we train a model on a training set, where the inputs are perturbed by adding a small amount of noise generated by the NSGA-II optimization-based perturbation method to increase the uncertainty of the test input for the DL model. Algorithm 2 shows the proposed method for adversarial training. The algorithm's input is the dataset D, the number of iterations I, the learning rate α, the model F, and the loss function L. The output of the algorithm is the robust model F'.

Algorithm 2 Adversarial Training

Require: D, I, α, F, L
Ensure: F'
1: Initialize the parameters of F;
2: Initialize the parameters of F';
3: **for** $i = 1$ to I **do**
4: Randomly select an input x_i from D;
5: $x_{unc} := x_i + \alpha \cdot NSGA - II(x_i, y_i, F)$;
6: Update the model F by minimizing $L(x_{unc}, y_i, F)$;
7: Update the model F' by minimizing $L(x_{adv}, y_i, F')$;
8: **end for**
9: **return** F'

5. Experimental evaluation

The aim of our experiments is to assess the effectiveness of our approach in minimizing the effect of uncertainty in the prediction results. The experimental setup and conditions are chosen to ensure that the results are as close to real-world conditions as possible. The results are then used to improve the model, and the process is repeated until the desired level of accuracy is achieved. We determined the best hyper-parameters for the DNN models for the datasets used in the experiments using a simple grid search. We found that the best hyper-parameters are the **ADAM** optimization with a learning rate of 0.001. Fig. 3 shows the DNN model. Dropout layers have been used during the model's training against adversarial test input injection. ReLu activation is used in the input and all the hidden layers; the output layer is activated by the Sigmoid function with the loss given by categorical cross-entropy. We perform the optimization using the Adam optimizer. The data is available from.[1] We present information about the dataset we employed in the experiments in Section 5.2, followed by the training process description.

[1] https://level-5.global/data/perception/.

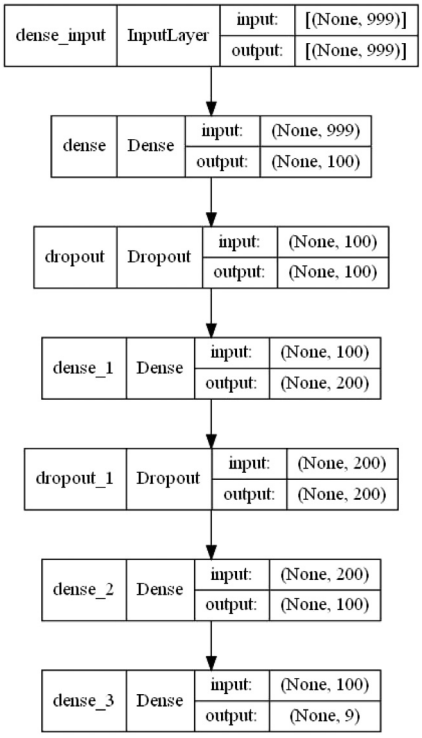

Fig. 3. Architecture of the deep neural network.

5.1. Generation of challenging inputs based on uncertainty maximization

Our approach is based on generating challenging inputs based on uncertainty maximization. Our method is similar to the methods utilized for defending against adversarial ML attacks. Here, we try to increase the prediction uncertainty of the model by adding a small perturbation to the input. In adversarial ML attacks, the objective is to fool the model using the loss maximization approach. We extend this approach by maximizing the prediction uncertainty in the model's prediction. We depict the overview of our methodology in Fig. 4. In the first step, the data is preprocessed including imbalance correction and train-test split. The second step trains the deep neural network model is trained on the train set. Once the model has sufficient performance, an adversarial test input generation is used to modify the training dataset. In the third step, the model is re-trained on the adversarial dataset. The model becomes more robust to this type of adversarial attack based on our approach.

To generate challenging inputs, i.e., highly uncertain inputs, we used an optimization method to find the best perturbation points on the input to increase the uncertainty. The objective function of the optimization method is to maximize the prediction uncertainty. There are three parts to the objective function, increase uncertainty, lower perturbation amount, and increase wrong prediction amount. The objective function combines the hyper-parameters (α and β) to find the optimal solution. The optimization objective is to create new instances based on training data with the following optimization function given in Eq. (5). Eq. (5) is constrained by the noise budget ϵ: $\|\mathbf{noise}\| < \epsilon$. We have assigned ϵ as 30% of the training dataset.

$$\mathbf{noise} = \underset{\mathbf{noise}}{arg\ min} \left(\frac{\alpha}{\mathbb{U}(h(\mathbf{x}))} + \|\mathbf{noise}\| + \beta \cdot \mathbb{1}(y == \hat{y}) \right), \quad \|\mathbf{noise}\| < \epsilon \tag{5}$$

where:

– $\mathbb{U}(h(\mathbf{x}))$ is the prediction uncertainty
– $\mathbf{x}$ is the input instance
– h is the DL model
– $\|\mathbf{noise}\|$ is the norm (i.e. magnitude) value of the noise
– ϵ is the Noise budget (i.e. the maximum distance between $\mathbf{x}$ and $\mathbf{x} + \mathbf{noise}$)
– $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{noise} \in \mathbb{R}^m$
– $\mathbf{noise}$ has the same number of columns as the input instance $\mathbf{x}$
– α is the multiplication factor of uncertainty quantification metrics value
– β is the multiplication factor of the number of incorrect predictions

The ideal **noise** would have the following properties:
– Maximize the prediction uncertainty: $\frac{\epsilon}{\mathbb{U}(\mathbf{x})}$
– Minimize the noise magnitude (norm value): $\|\mathbf{noise}\|$
– Change the prediction class: $\mathbb{1}(y == \hat{y})$

We use the generated data to train the model to increase its robustness.

5.2. Dataset

We use the LIDAR data in this study published by Woven Planet Holdings [36]. The data is available from https://level-5.global/data/perception. The LIDAR dataset size is about 37 GB. The dataset is collected as cloud points as shown in Fig. 5. The cloud points are represented as 3D-coordinate points such as (x, y, z) where x, y, and z are floats. In this dataset, a LIDAR scene corresponds to 4 images. However, we do not consider images in this study. The reason for using LIDAR data as opposed to images is that LIDAR sensors, unlike cameras, function independently of ambient lighting. LIDAR can achieve great results both day and night without any loss of performance due to disturbances such as headlight glare, sunlight, shadows, or lack of ambient lighting. The LIDAR sensors use an eye-safe laser to emit light pulses that light up the region of interest.

This dataset was initially released for regression models. However, we modified the dataset for training classification models. Since the dataset is too large, we down-sampled to 10% of the original dataset. The LIDAR dataset contains labels of the objects. We cropped the dataset and obtained the objects and their labels. The new dataset consists of only LIDAR data of the objects and their labels. There are 9 types of objects: animals (anml), bicycle (bcycl), bus, car, emergency vehicle (EV), motorcycle (MC), other vehicle (OV), pedestrian (pdstrn), and truck (trck). Only these objects are extracted to prepare the train and test sets. There is a class imbalance issue as there are more cars than cyclists or emergency vehicles. Therefore, we also applied class imbalance correction. The minority classes are oversampled through imbalance correction and the majority (car) class is undersampled. The training dataset is 67% and the testing dataset is 33%. The ML training is initiated by training on LIDAR data and labels. The model outputs the *softmax* classification of the objects. Then, using the uncertainty formula and NSGA-II algorithm, the dataset is modified with noise and sent to the ML models again for re-training.

5.3. Training

For defining the models and visualizing the analysis using Python, the following packages are used: TensorFlow, uncertainty wizard, Pandas, NumPy, Sci-Kit learn, SciPy, Keras, Seaborn, Pymoo, CleverHans, Tqdm, and Matplotlib.

The model is defined as a Stochastic Sequential Keras model with dense and dropout layers. ReLu activations are used for input and hidden layers with softmax activation in the final output. The

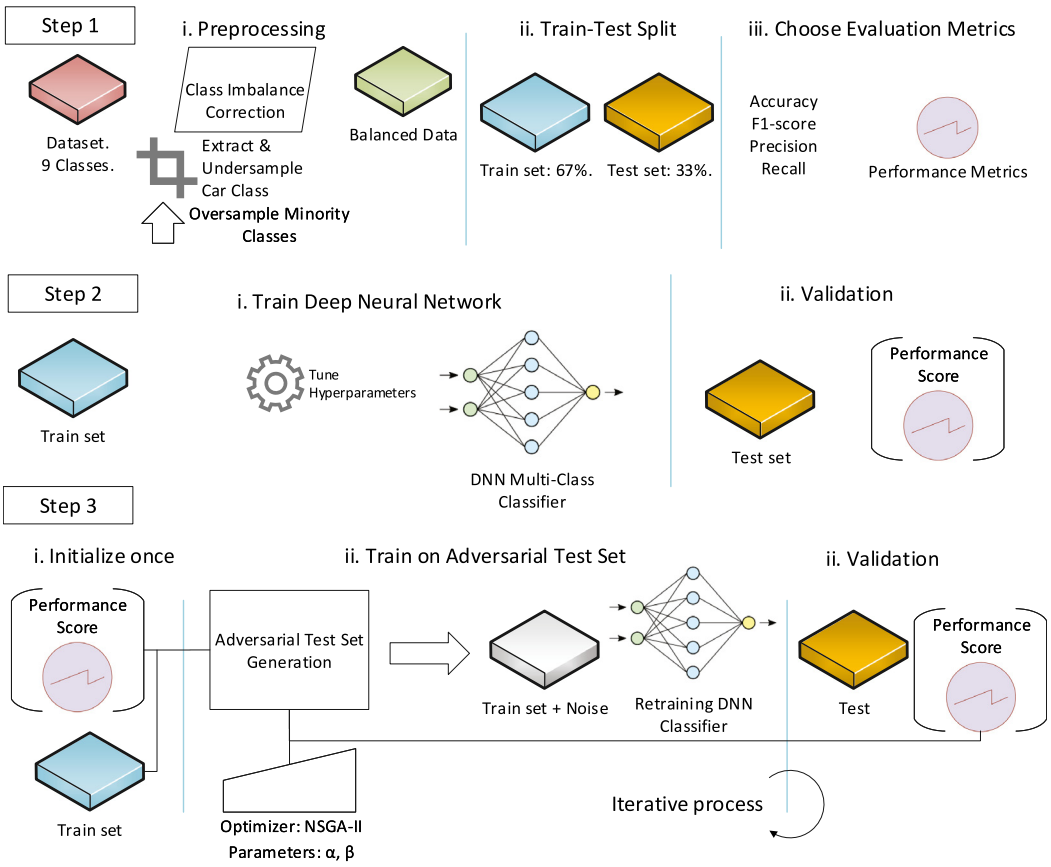

Fig. 4. Steps of the uncertainty-based test input generation and DL model training.

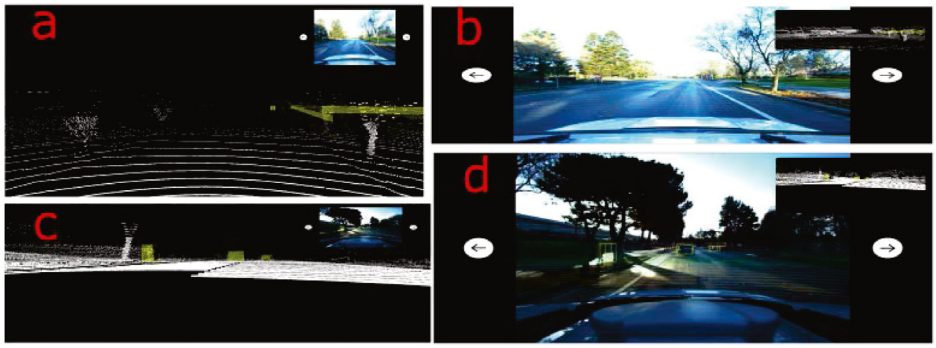

Fig. 5. In this figure, (a) shows a LIDAR output, and its corresponding image in visible light is shown in (b). (c) is the LIDAR output of a different scene, and the corresponding visible light image is shown in (d).

neurons in the dense layers are 100, 200, and 100, respectively. The dropout layers use 9% dropout. Adam optimizer is used with the loss function as categorical cross-entropy. The model is executed and trained with early callbacks enabled.

A different type of neural network, Pointnet, is also used for validating our approach. Pointnet is a neural network designed to work directly with point clouds while considering the permutation in-variance. [37].

6. Results and discussions

The performance of the ML model is evaluated using accuracy (defined in Eq. (6)), F-1 score (defined in Eq. (9)), recall, and precision. These metrics are determined using True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). The F1-score is calculated based on precision and recall and a value of the F1-score close to 1 indicates that the model performs well in both precision (see Eq. (7)) and recall (see Eq. (8)).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6}$$

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

$$F1\text{–}score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{9}$$

Fig. 6 plots losses and accuracy as the model is trained against the adversarial test input injection dataset. We see that accuracy is improving with a reduction in losses. Fig. 7 shows the accuracy, precision, recall, and F1 score during the adversarial training. The

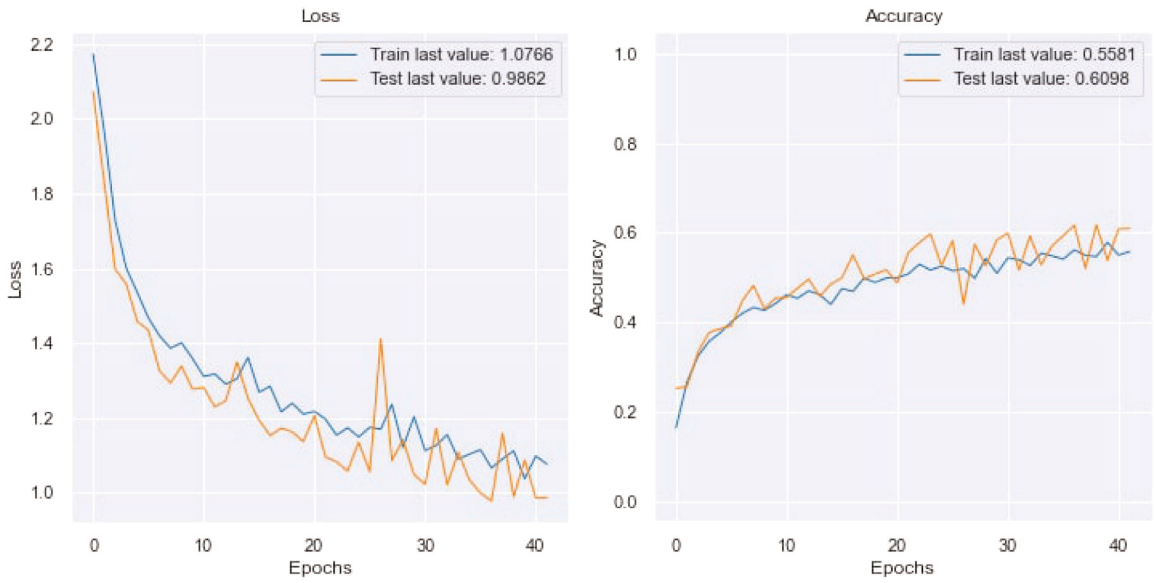

Fig. 6. Loss plot training.

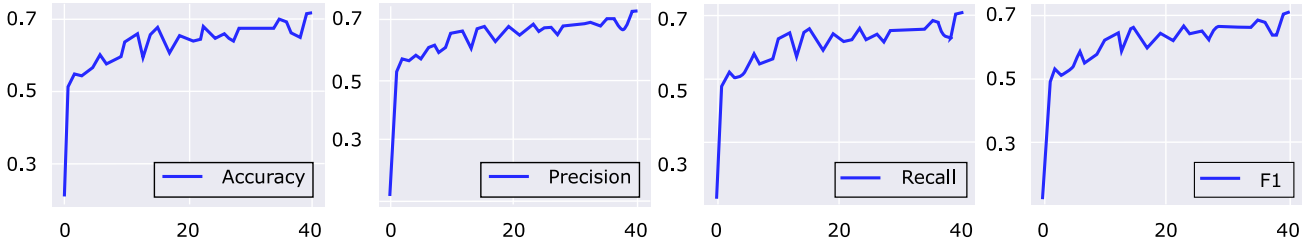

Fig. 7. Accuracy, precision, and recall with adversarial test input injection training. Model performance improves with iterations.

performance metrics are improving with every training iteration. Fig. 8 shows the drop in uncertainty before and after training with the adversarial inputs.

The model is made more robust with adversarial ML-based test input generation. After we append the dataset with the adversarial inputs by the allocated percentage of the noise budget, we re-train our model. In Fig. 7 we observe the model performance drops initially but improves again and becomes more robust with subsequent training iterations. Fig. 8 shows the drop in uncertainty after training with adversarial test inputs. Fig. 9 shows the improvement of the model across all the performance metrics after training using the adversarial test input set. The Pointnet performs better than the typical dense neural network as the architecture of this neural network is specially designed to directly handle point clouds.

Table 2 shows the influence of α and β hyper-parameters on the performance of the DNN during the training against adversarial test input injection. α is the multiplication factor of uncertainty quantification metrics value. We used mean–variance as the uncertainty quantification metric. β is the multiplication factor of the number of incorrect predictions. Varying these hyper-parameters will determine how much of the noise originates from the magnitude of the prediction error, or the number of wrong predictions. The hyperparameter values depend on the type of data and the classification task. Our optimization function uses these two hyper-parameters to penalize the classifier and train the model to become robust to adversarial test input injection. The best value for α and β is 0.6. The model showed a relatively optimal performance using this value which yields an average accuracy of 0.73, i.e., a 12% improvement upon the base model. The accuracy of the base model is 0.62.

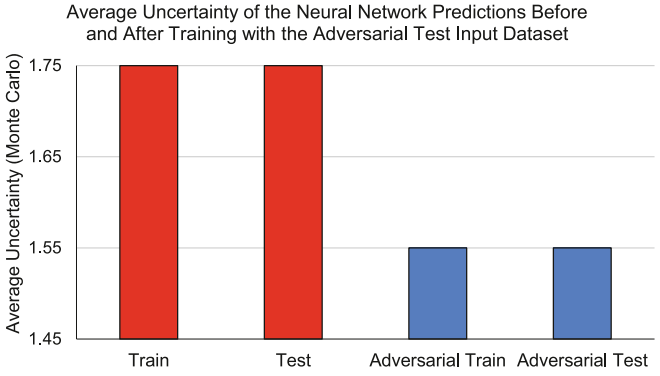

Fig. 8. Average uncertainty of the neural network predictions before and after training on the dataset with the adversarial test inputs. Average uncertainty drops after training on the adversarial dataset.

Table 3 shows the class-wise prediction performance scores for the robust model. The accuracy, precision, recall, and F1-score of the model show modest performance in absolute terms. The robust model exhibits relatively better performance than the base model and is resistant to adversarial test input attacks.

In the context of autonomous driving, these results indicate that robust models would be better at correctly classifying the objects and will be secure against adversarial attacks with test input generation when compared against a base model that has not been trained likewise. The results also show that there is room for improvement across all the metrics. Nevertheless, the

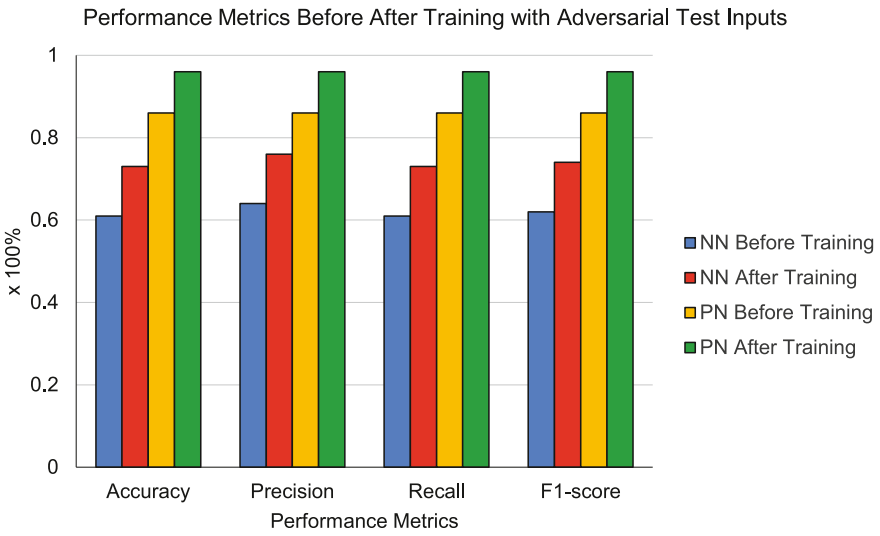

Fig. 9. The performance of the model improves across all metrics after training with adversarial test inputs. NN=dense neural network, PN=Pointnet.

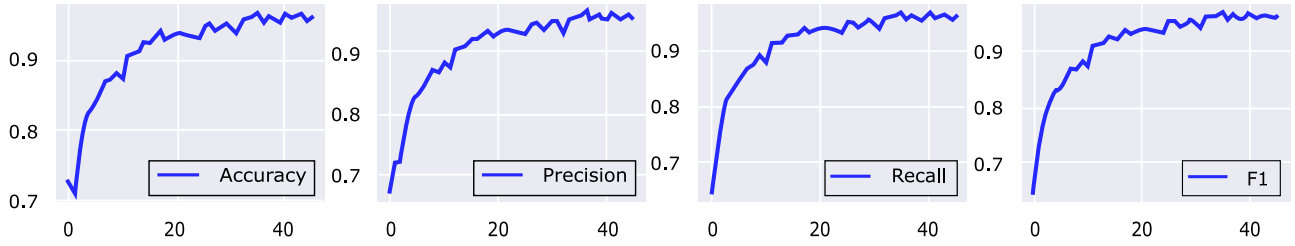

Fig. 10. Accuracy, precision, and recall with adversarial test input injection training. the Pointnet model, performance improves with iterations.

Table 2
The table shows the values of the α and β parameters and the resulting model performance, the % improvement compared to the base model, and the average performance metrics.

α	β	Accuracy	Precision	Recall	F1-score	α/β	Acc. improved
0.9	0.9	0.68	0.69	0.68	0.68	1	7.3
0.9	0.6	0.67	0.67	0.68	0.67	1.5	5.9
0.9	0.3	0.69	0.70	0.68	0.69	3	8
0.9	0.1	0.62	0.61	0.63	0.62	9	1.53
0.6	0.9	0.68	0.69	0.67	0.68	0.67	7.33
0.6	0.6	0.73	0.76	0.73	0.74	1	12
0.6	0.3	0.61	0.59	0.61	0.60	2	−0.27
0.6	0.1	0.65	0.66	0.64	0.65	6	4.06
0.3	0.9	0.60	0.62	0.61	0.61	0.33	−0.59
0.3	0.6	0.71	0.71	0.70	0.70	0.5	9.74
0.3	0.3	0.60	0.60	0.61	0.60	1	−0.64
0.3	0.1	0.69	0.70	0.68	0.69	3	8.16
0.1	0.9	0.72	0.71	0.70	0.70	0.11	11
0.1	0.6	0.63	0.62	0.61	0.61	0.17	2.01
0.1	0.3	0.52	0.52	0.51	0.51	0.33	−9.36
0.1	0.1	0.61	0.60	0.61	0.60	1	0.08

Table 3
The class-wise performance metrics of the DL model with the optimal α and β trained against adversarial test input injection dataset.

	Animal	bcycl	Bus	Car	EV	MC	OV	pdstrn	trck
Accuracy	0.76	0.74	0.73	0.72	0.73	0.73	0.71	0.70	0.64
Precision	0.74	0.73	0.74	0.71	0.73	0.74	0.71	0.70	0.61
Recall	0.77	0.74	0.75	0.72	0.71	0.75	0.73	0.68	0.62
F1-score	0.75	0.73	0.74	0.71	0.72	0.74	0.72	0.69	0.61

robust model is already performing better than the base model by a margin of 12%. Pointnet is used to classify point cloud data for the purpose of testing the adversarial test input injection training with a better-suited model. The resulting model without the input injection the model had an accuracy of around 85%; after the injection, the accuracy and other metrics reached around 96%. The improvement of 11% indicated that the method could work on diverse model types. The performance of the trained robust Pointnet model across various metrics is shown in Fig. 10.

7. Conclusion and future works

Adversarial attacks are used maliciously to confuse ML models. Uncertainty, which is inherent in the training data, or arising from the training process, has a similar effect on the performance of ML models. This can have severe consequences in the case of autonomous vehicles where correct identification of objects in the surroundings and road is critical. In this work, we used an adversarial test input generation approach for making the ML model more robust against uncertainty and adversarial attacks. We demonstrated that adversarial test inputs can affect the performance of the ML model, which can have severe consequences, such as the mis-classification of objects on the road by autonomous vehicles. However, by building a dataset that contains adversarial test inputs and re-training, we can make the ML model more robust to adversarial attacks. The accuracy of the robust model can reach higher than that of the base model by up to 12%, with a scalar reduction of 0.25 in uncertainty.

Our main contribution is a new method based on the NGSA-II algorithm for generating highly uncertain test input data for a given DL model for autonomous driving systems. We increased the robustness of the DL models against high uncertainty using the adversarial training mitigation method. The presented approach performs better in producing robust DL models against similar works in the literature.

In future work, we plan to address data uncertainty problems in addition to the model uncertainty problem addressed in this article. For this purpose, we aim to investigate Kernel Density Estimation (KDE) based methods for moving the data instances in a training set into low-density regions, thus increasing the uncertainty of the training data. Another improvement to the current approach of generating uncertain test data is looking at a DL loss function-based method for generating data instances with high uncertainty. The NGSA-II based optimization used in this work exhibits good performance. However, NGSA-II takes a long time to generate highly uncertain examples. The DL loss function-based uncertainty maximization method has a similar performance with lower execution times. Therefore, we will investigate this method in our future works.

CRediT authorship contribution statement

Devrim Unal: Conceptualization, Introduction, Related works, Formal analysis, Conclusion, Writing – original draft. **Ferhat Ozgur Catak:** Conceptualization, Literature review, Coding, Methodology, Review of draft. **Mohammad Talal Houkan:** Training ML models, Results, Discussion, Review of draft. **Mohammed Mudassir:** Methodology, Experiments, Discussion, Conclusion, Review of draft. **Mohammad Hammoudeh:** Conceptualization, Methodology, Discussion, Validation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Popoola SI, Adebisi B, Ande R, Hammoudeh M, Anoh K, Atayero AA. SMOTE-DRNN: A deep learning algorithm for botnet detection in the internet-of-things networks. Sensors 2021;21(9).

[2] Popoola SI, Adebisi B, Hammoudeh M, Gacanin H, Gui G. Stacked recurrent neural network for botnet detection in smart homes. Comput Electr Eng 2021;92:107039.

[3] Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L. ImageNet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. 2009, p. 248–55. http://dx.doi.org/10.1109/CVPR.2009.5206848.

[4] Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, et al. Imagenet large scale visual recognition challenge. Int J Comput Vis 2015;115(3):211–52.

[5] Jamil MA, Arif M, Abubakar NSA, Ahmad A. Software testing techniques: A literature review. In: 2016 6th international conference on information and communication technology for the muslim world (ICT4M). 2016, p. 177–82. http://dx.doi.org/10.1109/ICT4M.2016.045.

[6] Hamza Z, Hammad M. Testing approaches for web and mobile applications: An overview. Int J Comput Digit Syst 2020;9(4):657–64.

[7] Qayyum A, Usama M, Qadir J, Al-Fuqaha A. Securing connected & autonomous vehicles: Challenges posed by adversarial machine learning and the way forward. IEEE Commun Surv Tutor 2020;22(2):998–1026.

[8] Wickramasinghe CS, Marino DL, Amarasinghe K, Manic M. Generalization of deep learning for cyber-physical system security: A survey. In: IECON 2018 - 44th annual conference of the IEEE industrial electronics society. 2018, p. 745–51. http://dx.doi.org/10.1109/IECON.2018.8591773.

[9] Catak F, Yue T, Ali S. Prediction surface uncertainty quantification in object detection models for autonomous driving. In: 2021 IEEE international conference on artificial intelligence testing (AITest). Los Alamitos, CA, USA: IEEE Computer Society; 2021, p. 93–100. http://dx.doi.org/10.1109/AITEST52744.2021.00027, URL https://doi.ieeecomputersociety.org/10.1109/AITEST52744.2021.00027.

[10] Catak FO, Yue T, Ali S. Uncertainty-aware prediction validator in deep learning models for cyber-physical system data. ACM Trans Softw Eng Methodol 2022.

[11] Popoola SI, Ande R, Adebisi B, Gui G, Hammoudeh M, Jogunola O. Federated deep learning for zero-day botnet attack detection in IoT-edge devices. IEEE Internet Things J 2022;9(5):3930–44.

[12] Geisberger E, Broy M, editors. Living in a networked world: Integrated research agenda cyber-physical systems (agendaCPS). acatech Studie, München: Utz; 2015, URL http://www.acatech.de/de/publikationen/empfehlungen/acatech/detail/artikel/living-in-a-networked-world-integrated-research-agenda-cyber-physical-systems-agendacps.html.

[13] Zhang M, Ali S, Yue T, Norgren R, Okariz O. Uncertainty-wise cyber-physical system test modeling. Softw Syst Model 2019;18(2):1379–418.

[14] Ma T, Ali S, Yue T. Testing self-healing cyber-physical systems under uncertainty with reinforcement learning: an empirical study. Empir Softw Eng 2021;26(3):1–54.

[15] Amanullah MA, Habeeb RAA, Nasaruddin FH, Gani A, Ahmed E, Nainar ASM, Akim NM, Imran M. Deep learning and big data technologies for IoT security. 2020, http://dx.doi.org/10.1016/j.comcom.2020.01.016.

[16] Boukerche A, Ma X. Vision-based autonomous vehicle recognition: A new challenge for deep learning-based systems. ACM Comput Surv 2021;54(4).

[17] Khan IA, Moustafa N, Pi D, Haider W, Li B, Jolfaei A. An enhanced multi-stage deep learning framework for detecting malicious activities from autonomous vehicles. IEEE Trans Intell Transp Syst 2021;1–10.

[18] Jebamikyous H-H, Kashef R. Autonomous vehicles perception (AVP) using deep learning: Modeling, assessment, and challenges. IEEE Access 2022;10:10523–35.

[19] Zhu Q, Li W, Kim H, Xiang Y, Wardega K, Wang Z, Wang Y, Liang H, Huang C, Fan J, Choi H. Know the unknowns: Addressing disturbances and uncertainties in autonomous systems. In: Proceedings of the 39th international conference on computer-aided design. New York, NY, USA: Association for Computing Machinery; 2020, http://dx.doi.org/10.1145/3400302.3415768.

[20] Ben Lakhal NM, Adouane L, Nasri O, Ben Hadj Slama J. Safe and adaptive autonomous navigation under uncertainty based on sequential waypoints and reachability analysis. Robot Auton Syst 2022;152:104065.

[21] Michelmore R, Wicker M, Laurenti L, Cardelli L, Gal Y, Kwiatkowska M. Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control. 2020,

[22] Lakshminarayanan B, Pritzel A, Blundell C. Simple and scalable predictive uncertainty estimation using deep ensembles. In: Proceedings of the 31st international conference on neural information processing systems. Red Hook, NY, USA: Curran Associates Inc.; 2017, p. 6405–16.

[23] Ma W, Papadakis M, Tsakmalis A, Cordy M, Traon YL. Test selection for deep learning systems. ACM Trans Softw Eng Methodol 2021;30(2).

[24] Pei K, Cao Y, Yang J, Jana S. DeepXplore: Automated whitebox testing of deep learning systems. Commun ACM 2019;62(11):137–45.

[25] Tian Y, Pei K, Jana S, Ray B. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In: Proceedings of the 40th international conference on software engineering. 2018, p. 303–14.

[26] Ma L, Zhang F, Sun J, Xue M, Li B, Juefei-Xu F, Xie C, Li L, Liu Y, Zhao J, et al. Deepmutation: Mutation testing of deep learning systems. In: 2018 IEEE 29th international symposium on software reliability engineering (ISSRE). IEEE; 2018, p. 100–11.

[27] Ma L, Juefei-Xu F, Xue M, Li B, Li L, Liu Y, Zhao J. Deepct: Tomographic combinatorial testing for deep learning systems. In: 2019 IEEE 26th international conference on software analysis, evolution and reengineering (SANER). IEEE; 2019, p. 614–8.

[28] Xie X, Ma L, Juefei-Xu F, Chen H, Xue M, Li B, Liu Y, Zhao J, Yin J, See S. Deephunter: Hunting deep neural network defects via coverage-guided fuzzing. 2018, arXiv preprint arXiv:1809.01266.

[29] Ma W, Papadakis M, Tsakmalis A, Cordy M, Traon YL. Test selection for deep learning systems. ACM Trans Softw Eng Methodol 2021;30(2).

[30] Djenouri Y, Belhadi A, Srivastava G, Djenouri D, Lin JC-W. Vehicle detection using improved region convolution neural network for accident prevention in smart roads. Pattern Recognit Lett 2022;158:42–7.

[31] Mekala M, Park W, Dhiman G, Srivastava G, Park JH, Jung H-Y. Deep learning inspired object consolidation approaches using lidar data for autonomous driving: a review. Arch Comput Methods Eng 2021;1–21.

[32] Hüllermeier E, Waegeman W. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. Mach Learn 2021;110(3):457–506.

[33] Gane A, Hazan T, Jaakkola T. Learning with Maximum A-Posteriori Perturbation Models. In: Kaski S, Corander J, editors. Proceedings of the seventeenth international conference on artificial intelligence and statistics. Proceedings of machine learning research, Vol. 33, Reykjavik, Iceland: PMLR; 2014, p. 247–56, URL https://proceedings.mlr.press/v33/gane14.html.

[34] Maddox W, Garipov T, Izmailov P, Vetrov DP, Wilson AG. A simple baseline for Bayesian uncertainty in deep learning. 2019, CoRR, abs/1902.02476 arXiv:1902.02476.

[35] Deb K, Agrawal S, Pratap A, Meyarivan T. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: International conference on parallel problem solving from nature. Springer; 2000, p. 849–58.

[36] Kesten R, Usman M, Houston J, Pandya T, Nadhamuni K, Ferreira A, Yuan M, Low B, Jain A, Ondruska P, Omari S, Shah S, Kulkarni A, Kazakova A, Tao C, Platinsky L, Jiang W, Shet V. Level 5 perception dataset 2020. 2019, https://level-5.global/level5/data/.

[37] Qi CR, Su H, Mo K, Guibas LJ. Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2017, p. 652–60.