FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Attacking an Autonomous Vehicle Brake Anomaly Detector with Adversarial Learning Techniques

Francisco Maria Fernandes Machado Santos

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. Ricardo Morla

December 18, 2020

Attacking an Autonomous Vehicle Brake Anomaly Detector with Adversarial Learning Techniques

Francisco Maria Fernandes Machado Santos

Mestrado Integrado em Engenharia Informática e Computação

December 18, 2020

Abstract

Fully-automated vehicles are closely becoming more than a theoretical concept, result of the exponential interest demonstrated by the modern day society as well as the numerous advances achieved by the research community. The security of these intelligent systems is imperative for guaranteeing a safe and coordinated environment for these to operate and consequently integrate with our daily lives.

This work presents a study on the development of an autonomous vehicle brake-specific anomaly detection model using machine learning techniques as well as its vulnerabilities when exposed to state-of-the-art attacks, known as adversarial learning. The hyper-realistic Carla simulator together with the ROS framework are used for conducting simulated experiments, and a novel component is proposed for integration of anomaly detection machine learning models with autonomous driving simulations. We show that the proposed anomaly detection model can be trained in simulation under realistic conditions to yield high detection rates, and that this same model can be attacked resulting in drastic detection rate degradation.

Keywords: Autonomous Vehicles, Anomaly Detection, Adversarial Learning, Carla Simulator, ROS, One-Class Classification, Binary Classification, Multi-Class Classification, Deep Learning

Resumo

Os veículos totalmente autónomos estão a tornar-se mais do que um simples conceito teórico, como consequência dos inúmeros avanços alcançados pela comunidade científica bem como do interesse demonstrado pela sociedade moderna. A segurança destes sistemas inteligentes é imperativo para garantir um ambiente seguro e coordenado onde estes possam operar e, consequentemente, integrar o nosso quotidiano.

Este trabalho apresenta um estudo do desenvolvimento de um modelo específico de deteção de travagens anómalas de um veículo autónomo utilizando técnicas de *machine learning* bem como a exposição das suas vulnerabilidades quando exposto a ataques de última geração, conhecidos como *adversarial learning*. A utilização do simulador hiper-realista, Carla simulator, juntamente com a *framework* ROS, permitiram a realização de experiências simuladas, e um novo componente é proposto para permitir a integração de modelos de *machine learning* para deteção de anomalias com as simulações de condução autónoma. Apresentamos o treino do modelo de deteção de anomalias em simulação, sob condições realistícas, alcançando uma elevada taxa de deteção, e, ainda, que este mesmo modelo pode ser alvo de ataque resultando numa redução drástica da sua taxa de deteção.

Keywords: Autonomous Vehicles, Anomaly Detection, Adversarial Learning, Carla Simulator, ROS, One-Class Classification, Binary Classification, Multi-Class Classification, Deep Learning

Acknowledgements

À minha Mãe, Anabela, agradeço todo o carinho, força, dedicação e amor que sempre me demonstrou e que guardo com o meu maior orgulho. Por toda a sua amizade e compreensão que me permitiram ambicionar e atingir etapas que não seriam alcancáveis sem o seu apoio. Sou eternamente grato por todo o seu amor e carinho e farei tudo o que estiver ao meu alcance para que sempre se orgulhe de mim.

Aos meus Avós, Maria Delfina e Norberto, quem eu profundamente admiro, por todo o amor e carinho bem como todos os ensinamentos, conselhos e vivências que jamais irei esquecer. Por serem a minha maior fonte de inspiração. Dedico e dedicar-lhes-ei todos os meus sucessos. Embora já não presentes fisicamente, estarão sempre comigo no meu Coração.

Ao meu Pai, Pedro, pelo seu apoio, força e motivação que demonstrou durante este capítulo essencial do meu percurso académico.

À minha Namorada, Helena Beatriz, por todo o amor, carinho, apoio e motivação que sempre me demonstrou, nunca deixando de confiar nem de acreditar em mim, tendo estado a meu lado nas etapas mais importantes da minha vida e que sem o seu apoio e amor seriam impossíveis de ultrapassar. Por todo o amor incondicional que nos une e que, juntos, nos permite ultrapassar qualquer obstáculo. Por me mostrar o lado mais belo e feliz da Vida e me dar o privilégio de a passar a seu lado.

Aos meus Amigos, por estarem sempre comigo e por todo o apoio e força que sempre me dedicaram. Por tudo aquilo que passámos e vivemos. Por toda a nossa amizade que me faz orgulhar de os poder chamar de Irmãos. Não poderia ter pedido melhores pessoas para me acompanharem neste percurso que é a Vida.

A todos aqueles com quem tive o prazer de trabalhar e que me ajudaram durante o meu percurso Académico, Pessoal e Profissional, inclusive ao meu manager e colegas de estágio que foram essenciais para o meu desenvolvimento profissional e pessoal.

Um especial agradecimento ao Professor Ricardo Morla por toda a sua disponibilidade, compreensão, apoio e amizade que demonstrou desde o início deste trabalho e que sem a sua valiosa ajuda este não existiria. Por todos os esforços que fez para que este projeto fosse realizável bem como todo o apoio disponibilizado a qualquer hora. Recordá-lo-ei sempre com a maior admiração, como um excelente Professor e Amigo, pela sua dedicação, amizade e persistência.

Agradeço ainda a todo o corpo docente da FEUP que me acompanhou durante o meu percurso académico e que me permitiu adquirir o nível de conhecimento que hoje possuo.

A todos, dedico este trabalho. Obrigado por tudo.

Francisco Maria Fernandes Machado Santos

"Difficulties mastered are opportunities won."

Winston Churchill

Contents

1 Introduction **1**
- 1.1 Motivation . . . 2
- 1.2 Contributions . . . 2
- 1.3 Thesis Structure . . . 3

2 Technical Background & Literature Review **5**
- 2.1 Carla Simulator . . . 5
- 2.2 ROS . . . 7
  - 2.2.1 Core Concepts . . . 7
  - 2.2.2 Communication . . . 8
  - 2.2.3 Recording . . . 9
- 2.3 Carla-ROS Bridge . . . 9
- 2.4 Adversarial Learning . . . 9
  - 2.4.1 White-Box Adversarial Generation . . . 9
  - 2.4.2 Black-Box Adversarial Generation . . . 10
  - 2.4.3 Model Poisoning . . . 12
  - 2.4.4 Targets . . . 12
  - 2.4.5 Defenses . . . 13
- 2.5 Related Work . . . 13
  - 2.5.1 Machine Learning in Anomaly Detection . . . 13
  - 2.5.2 Adversarial Learning Applications . . . 15
  - 2.5.3 ROS Security Frameworks . . . 17
- 2.6 Summary . . . 17

3 Carla-ROS Bridge Anomaly Detection Component **19**
- 3.1 System Integration . . . 19
- 3.2 Architecture of the Component . . . 20
  - 3.2.1 Preprocessing Node . . . 20
  - 3.2.2 Preprocessed Data Topic . . . 21
  - 3.2.3 Anomaly Detector Node . . . 22
- 3.3 Summary . . . 26

4 Brake-Specific Anomaly Detection Model **29**
- 4.1 Methodology . . . 29
  - 4.1.1 Metrics . . . 30
  - 4.1.2 The Training and Primary Testing Phase . . . 31
  - 4.1.3 The Production Testing Phase . . . 33
- 4.2 Data Collection . . . 35

4.3 Data Preprocessing . . . 37
4.3.1 Parsing . . . 39
4.3.2 Data Labeling . . . 39
4.3.3 Handling Class Imbalance . . . 40
4.3.4 Normalization . . . 41
4.3.5 Feature Selection . . . 42
4.4 One-Class Classification . . . 43
4.4.1 Data Filtering . . . 43
4.4.2 Training and Primary Testing Phase . . . 44
4.4.3 Production Testing . . . 45
4.5 Binary Classification . . . 46
4.5.1 Model Architecture . . . 46
4.5.2 Data Labeling . . . 47
4.5.3 Class Imbalance . . . 48
4.5.4 Training and Primary Testing Phase . . . 49
4.5.5 Production Testing . . . 50
4.6 Multi-Class Classification . . . 51
4.6.1 Data Re-Labeling . . . 51
4.6.2 Class Imbalance . . . 52
4.6.3 Model Architecture . . . 52
4.6.4 Training and Primary Testing Phase . . . 53
4.6.5 Production Testing . . . 54
4.7 Comparative Analysis . . . 56
4.8 Model Selection . . . 58

5 Attacking the Anomaly Detection Component **61**
5.1 Adversarial Samples Generation . . . 61
5.2 Architecture of the Attack . . . 62
5.2.1 Description . . . 63
5.2.2 Practical Approach . . . 65
5.3 Experimental Setup & Observed Results . . . 65
5.3.1 No Anomaly Detection Run . . . 67
5.3.2 Attack-Free Anomaly Detection Run . . . 68
5.3.3 Adversarial Run . . . 69
5.4 Defenses . . . 70

6 Conclusions & Future Work **73**
6.1 Overview . . . 73
6.2 Future Work . . . 74

A Carla-ROS Bridge Anomaly Detection Component Usage **75**
A.1 Installation . . . 75
A.2 Usage . . . 76

B Brake Anomaly Detection Model Auxiliary Information **77**
B.1 Binary Classification Training Metrics . . . 77
B.2 Multi-Class Classification Training Metrics . . . 78
B.3 Features Exploratory Analysis . . . 79
B.4 Attack-Free Anomaly Detection Production Testing Results . . . 82

B.5 Attacked Anomaly Detection Production Testing Results 83

C Scripts **85**
C.1 Parsing Rosbag (.bag) files 85

References **89**

List of Figures

2.1 Carla simulator interaction window. 6
2.2 ROS communication diagram. 8
2.3 Carla-ROS bridge active topics upon a simulation run. 10

3.1 Diagram representing the entities and their interaction of the created anomaly detection component. 20
3.2 Fluxogram representing the workflow of the anomaly detection ROS node. 23
3.3 Diagram representing the entities and their interaction of the created anomaly detection component. 25
3.4 Anomaly detection component integration with Carla-ROS bridge. 27

4.1 Methodology followed for implementing, testing and comparing the different machine learning models. 30
4.2 Training and primary testing phase representation. 32
4.3 Workflow executed for each production testing simulation. 34
4.4 Location of the brakes in the simulated vehicle. 35
4.5 Diagram representing the active topics upon data collection. 38
4.6 Diagram representing the behavior of the script for parsing rosbag (`.bag`) files. . . 39
4.7 Label distribution transformation for the one-class classification model training data set. 44
4.8 Binary classification. 46
4.9 Architecture of the deep neural network created for binary classification. 47
4.10 Original class distribution for the binary classification model training data set. . . 48
4.11 Multi-class classification. 51
4.12 Class distribution for the multi-class classification model training data set. 52
4.13 Architecture of the deep neural network created for multi-class classification. . . 53
4.14 Relation between the model prediction and the velocity of the vehicle in an anomalous situation. 56
4.15 Charts of the metrics for comparison of the implemented approaches: one-class, binary and multi-class classification. 59

5.1 Adapted sequence diagram of the *Stealth Publisher Attack* to the proposed anomaly detection component. 64
5.2 Representation of the different simulations ran for testing the impact of the anomaly detector component. 66
5.3 No anomaly detection run chart 67
5.4 Attack-free anomaly detection run chart. 68
5.5 Adversarial run chart 69

B.1 Binary classification model training metrics. 77
B.2 Multi-class classification model training metrics. 78
B.3 Correlation matrix of the binary classification model training data set. 80
B.4 Correlation matrix of the multi-class classification model training data set. 81

List of Tables

3.1 Preprocessing Node topics interactions. 21
3.2 Preprocessing data topic information. 22
3.3 ad_predictions topic description. 24
3.4 Emergency mode state topic description. 25
3.5 Anomaly Detector Node topics interactions. 25

4.1 Confusion matrix. 31
4.2 Description of the brakes failing for each production testing simulation. 34
4.3 Topics selected for data collection. 36
4.4 Description of the scenarios used for data collection. 37
4.5 Description of the classes used for labeling the collected data. 40
4.6 Collected data set information. 40
4.7 Features selected for training of the machine learning models. 42
4.8 Original data set distribution for the construction of binary classification model. . 44
4.9 One-class classification model hyperparameters tuning impact on accuracy. . . . 44
4.10 One-class classification model RBF nu=0.4 production testing results. 45
4.11 One-class classification model RBF nu=0.4 production testing units results. . . . 45
4.12 Class weights specified for the target classes of the binary classification model. . 49
4.13 Original data set distribution for the construction of binary classification model. . 49
4.14 Binary classification model hyperparameters tuning. 49
4.15 Binary classification production testing results. 50
4.16 Binary classification production testing units results. 50
4.17 Description of the classes used for labeling the collected data. 52
4.18 Class weights specified for the target classes of the multi-class classification model. 52
4.19 Original data set distribution for the construction of binary classification model. . 53
4.20 Multi-class classification model hyperparameters tuning. 54
4.21 Multi-class classification production testing results. 54
4.22 Multi-class classification production testing units results. 54
4.23 Binary *versus* multi-class primary testing results comparison. 56

5.1 White-box attacks impact on the accuracy of the model. 62
5.2 Black-box attack impact on the accuracy of the model. 62
5.3 Description of the different simulations ran for testing the studied situations. . . . 65
5.4 Screenshots of the different experiments ran to asses the impact of both having a normal and exploited anomaly detection model. 71

A.1 Requirements for the successful installation of the anomaly detection component. 75

B.1 Features available in the data set of the collected data. 79

B.2 Results of the anomaly detection component predictions upon normal samples. . 82
B.3 Units results of the functional anomaly detection component. 82
B.4 Results of the anomaly detection component prediction upon adversarial samples. 83
B.5 Units results of the exploited anomaly detection component. 83

Abbreviations

ACC	Accuracy
ADM	Anomaly Detection Mechanism
AI	Artificial Intelligence
AV	Autonomous Vehicle
BIM	Basic Iterative Method
C&W	Carlini & Wagner
CFG	Control Flow Graph
CNN	Convolutional Neural Network
DoS	Denial of Service
DDoS	Distributed Denial of Service
DNN	Deep Neural Network
FGSM	Fast-Gradient Sign Method
FUT	Future Urban Transportation system
FP	False Positive
FN	False Negative
HTTP	Hypertext Transfer Protocol
ITS	Intelligent Transportation System
ML	Machine Learning
PPV	Positive Predicted Value
ROS	Robot Operating System
TCP	Transmission Control protocol
TPR	True Positive Rate
TP	True Positive
TN	True Negative

Chapter 1

Introduction

Autonomous driving has proven to be an increasingly popular subject mainly because of its expected future impact in our daily lives. Numerous technological advances originated by this exponential research interest have been produced, leading to the rapid development of autonomous vehicles (AVs) and its transverse subjects.

Currently, physical implementations of AVs already exist [56, 8] and are being tested in real-world scenarios. These complex systems combine state-of-the-art equipment, such as radars, cameras and sensors, with knowledge from different fields of research and areas of expertise [4]. It is expected that soon these vehicles will form a dense network of components, known as connected autonomous vehicles (CAVs), capable of communicating with each other in order to guarantee a safe and coordinated environment, aiming to reduce traffic accidents, improve quality-of-life and to enhance the efficiency of current transportation systems [27]. These components are expected to compose Intelligent Transportation Systems (ITS) which rely on a distributed architecture and an advanced communication infrastructure in which the user is the centre of all concerns. Research is being carried out towards the development and the implementation of Future Urban Transport systems (FUTs) which will have AVs as a key concept, where not only the user will be its central focus but also have a passive role, being managed by the system only [55].

Artificial Intelligence (AI) is the key concept common to these systems which provides the means for machines to learn how to efficiently execute a non-conventional task. Autonomous vehicles are AI-dependent systems, taking advantage of state-of-the-art machine learning techniques to be able to operate without any human intervention. Hence, the security of a single AV system is a critical and complex concept which not only affects its users and the system itself, but also all the external world entities that surrounds it - be that material property, animals or persons.

1.1 Motivation

Vehicles are susceptible to common mechanical problems which may threaten its well-functioning, possibly resulting in serious consequences depending on both its level of severity and the vehicle's surrounding conditions but also, for the great majority of current vehicles, the driver's response. On the other hand, fully-automated vehicles do not rely on a human entity for control. Therefore, such systems must be capable to automatically recognize such problems and efficiently decide on the safest option to mitigate them. Anomaly detection focuses on detecting uncommon behaviors that are assumed to be indicators of a compromise to a system's normal operation. The integration of components responsible for implementing anomaly detection in safety-critical systems such as AVs is essential to recognize any atypical scenarios and consequently contribute to the system's well-functional operation.

Lately, the evolution of machine learning techniques allowed the creation of anomaly detection models which have proven to obtain exceptional results when compared to other adopted techniques. However, such models are not bulletproof against attackers whose goal is to compromise their operation. Adversarial learning is a novel field of study which has proven to impose a strong and dangerous threat against current machine learning models by taking advantage of their underlying vulnerabilities through the generation of malicious inputs that are not able to be correctly identified and are thus able to deceive the model. When such techniques are applied to anomaly detection machine learning models, specifically those integrated in AV systems, their outcomes might be disastrous, compromising not only the AV's operation but the world that surrounds it.

1.2 Contributions

We expect to deliver a relevant contribution to the autonomous driving research community by presenting an in-depth study of the creation process of a brake-specific anomaly detection model using the realistic Carla simulator as basis to efficiently collect simulated information and later experiment the model in a simulated environment through a novel machine learning-based component. A thorough exposition of the model's development process is presented and three different machine learning approaches, based on both supervised and unsupervised learning techniques, are compared resulting in different solutions for the current problem. State-of-the-art adversarial learning attacks are applied to the created machine learning model under simulation, proving that these have the potential to negatively impact the vehicle's operation, ultimately causing an accident, and that defenses against them must be present in order to guarantee a safe and stable autonomous driving experience.

1.3 Thesis Structure

A novel anomaly detection component integrated with Carla-ROS bridge is presented in chapter 3, explaining its architecture and internal workflows. The creation of such component is expected to offer its users a well-structured and intuitive tool to integrate anomaly detection models within their simulated systems and thus experiment and hopefully contribute to new advances of machine learning anomaly detection systems in autonomous vehicles. This software project is designed to accept both contribution and maintenance by interested users who wish to expand or improve its current features.

Chapter 4 presents a thorough study on how a brake-specific anomaly detection model was constructed, unveiling the results of the different implemented approaches, using both supervised and unsupervised learning techniques.

At last, chapter 5 focuses on attacking the created anomaly detection model, presenting a ROS-specific attack as well as different adversarial learning techniques for generating samples to be maliciously injected into the anomaly detection model. Different experiments were conducted with the goal of understanding their impact and possible outcomes when applied against the brake-specific anomaly detection model.

Chapter 2

Technical Background & Literature Review

This chapter focuses on presenting the background of the tools used throughout the development of this work as well as a literature review of current state-of-the-art adversarial learning techniques against machine learning models, more specifically, anomaly detection models. The first sections introduce a brief overview of the used tools, explaining each core concepts, and also an explanation of what adversarial learning is and some of its most common techniques for generating adversarial examples. Finally, the last section of this chapter presents the literature review of this novelty type of machine learning attacks against modern anomaly detection models in the context of IoT and autonomous vehicles.

2.1 Carla Simulator

Autonomous driving simulation systems have seen an exponential interest accompanying the evolution of AI and autonomous driving. Such systems offer a secure and precise mechanism to test different autonomous driving features without the need of testing these in real physical vehicles and thus avoid risk of endangering the physical world that surrounds them, as well as to reduce the funds needed for such precise and complex testing.

Carla (Car Learning to Act) [24] is an open-source project which aims to assist autonomous driving research by providing hyper-realistic performance with state-of-the-art rendering quality and realistic behaviors. It supports a great variety of different configurations making it a flexible tool which offers a detailed specification support on sensor suites, environmental conditions and full control of the static and dynamic actors that are spawned during a simulation run. This simulator has a client-server architecture where the server is responsible for running the simulation and rendering the scene whereas the client interacts with it by sending commands to the server and

receiving sensor information in return. The latter is generated by objects responsible for retrieving specific data about the vehicle, known as sensors, and are defined in the simulation as actors themselves. Carla offers a great variety of sensors including collision and obstacle detectors, Lidar raycast and GNSS (*Global Navigation Satellite System*).

The simulation inside Carla has different synchronisation properties which can be altered by the user. The server, responsible for running the simulation, has its own clock and time where it tries to run the simulation as fast as possible, without waiting for the client, making it problematic for slow client applications which can not process the information returned by the server at a variable time-step. This causes an overflow of information which the client is incapable of processing resulting in either lost or mixed information, as well as the impossibility of knowing if all the sensors are using data from the same simulation moment. Therefore, synchronisation is essential so that both client and server share a fixed time-step, where the client has the responsibility of notifying the server whenever it is ready to process the next simulation step.

This simulator offers direct interaction with the vehicle through the input of keyboard controls in a game window, presented in figure 2.1, which will internally be converted to vehicle control commands. As such, Carla supports both manual and automatic control of the vehicle, also known as autopilot mode.

Figure 2.1: Carla simulator interaction window.

2.2 ROS

ROS, or *Robotic Operating System*, is an open-source software framework for developing robotics-specific software, being well-praised due to its notable features such as large systems support and language independence (i.e. multiple client libraries support). Current literature has presented different projects using ROS to implement a working autonomous vehicle system in urban environment [44].

At its core, ROS implements a publish/subscribe communication paradigm consisting on multiple entities, known as publishers and subscribers, capable of exchanging messages between them through a specific transport system known as topic, in both an asynchronous and anonymous way. In this many-to-many communication model, the publishing of a message to a topic will be broadcasted to all of its subscribers, without the publisher's knowledge of the receiving recipients. Such design model focuses on both performance and scalability, mainly targeting complex distributed systems.

2.2.1 Core Concepts

- Nodes: these entities are processes whose responsibility is to perform computations. Currently, the supported ROS client libraries allow the creation of such entities. Each node has a unique URI corresponding to its own XML-RPC server, used for negotiating connections with other nodes as well as to communicate with the Master. Multiple APIs, such as the *Slave API* are exposed so that other nodes including the master are able to communicate with it.

- Master: it is a special node in a ROS environment responsible for providing specific services to other nodes such as name registration and lookup methods to allow clients to find each other and thus start the exchange of messages between them. Such services make use of internal data structures responsible for collecting both publishers and subscribers information. Note that this node is implemented via XML-RPC permitting nodes to advertise its publishing intent and subscriber nodes to retrieve each of the publisher's location. Note that messages do not flow through this node as these are exchanged directly between the publisher and subscriber nodes.

- Topics: are responsible for transporting the messages from the publishers to the subscriber nodes. These are identified by a name and may be subscribed by multiple nodes. A topic supports a single messaging type for transfer of information.

- Message: these entities are not more than data structures representing information transferred from one node to another. Built-in messaging types (i.e. int and float) are available as well as the support for creation of custom message types.

2.2.2 Communication

As previously described, ROS allows different nodes to communicate with each other. However, its process is not trivial using different protocols, such as XML-RPC (lightweight stateless HTTP protocol) for successfully transferring information from one another. Figure 2.2 presents a visual representation of the initiation of the publishing of messages from one publisher node to a subscriber.

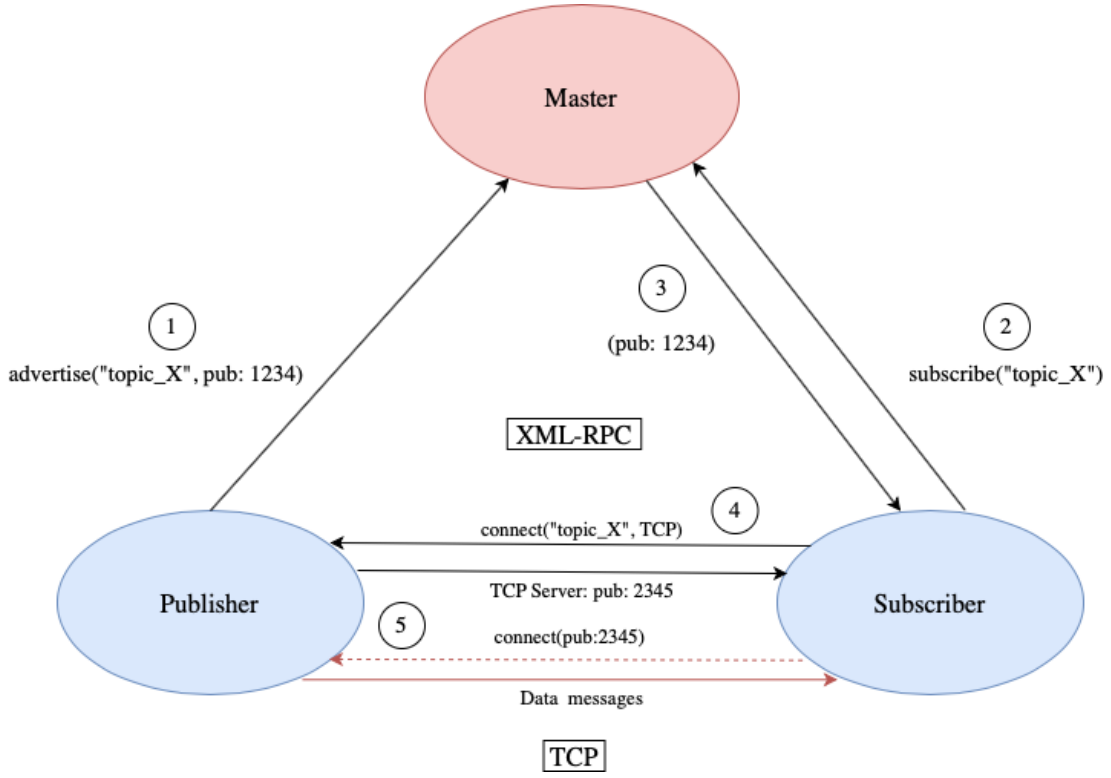


Figure 2.2: ROS communication diagram.

Whenever a publisher node wants to start publishing messages to a specific topic, it must first advertise its intent to the Master node, through the XML-RPC protocol, as represented by phase 1 in the figure. The Master, upon receiving the node's message stating its intent of publishing data to a specific topic, will look for any subscriber of the specified topic to check if there is an already active subscriber. If so, it will be contacted by the master using its *Slave API* stating that a new publisher of this topic is available.

A new subscriber node, must first communicate its intent of subscribing to a topic to the master via XML-RPC (phase 2 of the diagram), in which the latter will reply with a list of the XML-RPC servers of this topic's active publishers (phase 2 and 3 of the diagram, respectively). The subscriber will then initiate the negotiation of the communication protocol with each of the publishers and after successful agreement it is able to start receiving the published information (phase 4 of the diagram). A common protocol for communication between ROS nodes is TCPROS [70], consisting on standard TCP/IP sockets for the transport of messages. Inbound connections

must specify a connection header containing specific fields such as the message data type and the name of the topic.

2.2.3 Recording

ROS offers a built-in tool, known as *rosbag*, to record the published information flowing through the different topics. Such tool generates files of a specific format called bag (`.bag`) which allow the saving and later reproduction of the ROS messages data.

2.3 Carla-ROS Bridge

The Carla-ROS bridge software framework was design to allow two-way interaction between the Carla simulator and external entities, represented by ROS nodes. It receives information generated by the Carla server, such as sensor data or world details, and later publishes to topics for external access. On the other hand, it translates messages published to supported ROS topics, such as vehicle control commands, to Carla-specific controls which will affect the simulation.

This framework is extremely useful for both retrieving and sending information of the ongoing simulation from external entities to the Carla simulator, allowing different experiments to be conducted. Figure 2.3 presents a diagram of the active topics where simulation data is published to by the Carla-ROS bridge software stack.

2.4 Adversarial Learning

Adversarial Learning has been subject to an interesting research growth within the last years due to the exponential use and evolution of machine learning, currently almost transverse to any kind of application, including safety-critical systems such as AVs, and how this technique can compromise the security of systems relying on ML models to successfully operate.

At its core, evasion-based adversarial learning focuses on carefully crafting inputs to ML models so they can misclassify them and thus produce erroneous outputs. The generation of such inputs, known as adversarial examples or adversarial samples, is influenced by the adversary's ability to access the model's internals (architecture, parameter values or its training data). The generation of such inputs may be classified in two different adversarial example generation techniques regarding the adversary's strength against the model: *white-box* and *black-box* [29]. Another type of adversarial learning focuses on modifying the training data and it is called *model poisoning*.

2.4.1 White-Box Adversarial Generation

White-box techniques [30, 47, 26] are characterized by adversaries having knowledge on the target models internals. With this premise, numerous adversarial examples generation techniques were designed, proven to be very effective. The knowing of the models internals (parameters,

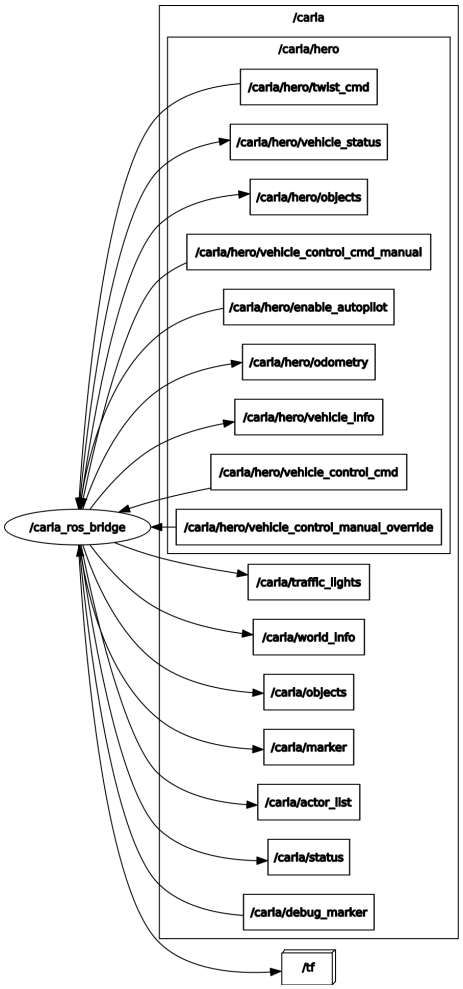


Figure 2.3: Carla-ROS bridge active topics upon a simulation run.

architecture, loss and activation functions, among others) allows the adversary to carefully craft adversarial examples specifically designed for the targeted model [78].

2.4.2 Black-Box Adversarial Generation

In most of real-world scenarios, adversaries do not have knowledge about the model's internals. We assume that the only way one can access and interact with these models is through a defined interface. This type of adversarial generation, and consequently attack, relies on the adversary's capability of accessing the labels assigned by the model, through the defined interface, for any chosen input. Therefore, the adversary relies on querying the model to retrieve the predicted labels. It is important to note that the number of queries made to a model is an important metric to evaluate the efficiency of the algorithm, due to performance costs and the risk of detection.

Research has found different methods to generate adversarial examples using a *black-box* approach. The nature of such methods can be divided into three different approaches [23].

- ***Transfer-based*** [19, 17, 53, 34, 43, 68]

  This solution is based on the transferability property of adversarial examples in machine learning models [37], which states that an adversarial example may be transferred from model to model and still remain adversarial even if the models' architecture varies greatly.

$$M_1(x') \neq y \tag{2.1}$$

$$M_2(x') \neq z \tag{2.2}$$

  Equations 2.1 and 2.2 present how the transferability property works by showing that the same input (x') to both models (M_1, M_2) is misclassified in both. This property raises severe security concerns for machine learning applications, as an adversary does not need to know the model internals to successfully craft adversary examples in a simple manner [17].

  Cheng et al. [19] presents a new method to improve the efficiency of exiting attacks, revealing improvements of 20% to 40% in some cases while comparing to other transfer-based methods. Papernot et al. [53] explored and concluded that some adversarial samples are misclassified across models trained with the same method and also across models trained with different techniques. Additionally, Liu et al. [43] proposes a new strategy to generate transferable adversarial images as well as the presentation of an extensive study on the transferability of different adversarial examples (*targeted* and *non-targeted*) over large data sets.

- ***Score-based*** [81, 3, 7]

  A score-based approach to generate adversarial examples relies on querying the model (DNN) to retrieve soft labels for the tested inputs. In other words, the adversary has only access to the score or probabilities predicted by a classifier for each class, for a given input.

  This approach, however, can not be generalized to most of real-world machine learning applications as the confidence scores are rarely accessible.

- ***Decision-based*** [54, 10, 16, 23, 18]

  This approach requires less knowledge on the target model as it only relies on the hard labels (final decision) returned by it. Although easier to apply when compared to the two previous approaches, this requires numerous queries to the model which may denounce its intentions.

  Brendel et al. [10] proposed a simple yet effective and flexible attack based on this approach: the *Boundary* attack. Such attack works by starting from a large perturbation and successfully reducing it, following the decision boundary between adversarial and non-adversarial samples.

It should be noted that these approaches are not mutually exclusive. In fact, most of the *black-box* methods combine some or all of these approaches. Papernot et al. [54] presents a

new, reverse-engineering, *black-box* adversarial generation technique, combining both *transfer* and *decision-based* approaches, which allows an adversary to generate malicious samples to attack a ML model, more specifically DNNs (Deep Neural Networks). This strategy consists in learning a substitute DNN approximating the target using a data set constructed with synthetic inputs and labels observed from the interaction with the targeted model's API. Adversarial examples are then crafted using this substitute and applied to the original model.

2.4.3 Model Poisoning

Also known as *false learning*, model poisoning consists on the *data poisoning* technique which focuses in manipulating the training phase of a machine learning model by injecting adversarial samples in the training so that the algorithm learns the wrong data. This allows an adversary to later attack the manipulated ML model successfully as it will fail to detect the adversarial examples.

Initially, these attacks focused on offline or batch settings where the adversary attacks the full set or a full batch of the set of the training data, however, Zhang et al. [79] presented a new approach to *data poisoning* based on sequential learners - models that are continuously receiving data. With it, the adversary acts as a *man-in-the-middle* and observes the training samples, the learner's model, and injects poisoned samples. Rubinstein et al. [60] presents an extensive study on the effects of multiple poisoning strategies which have proven to severely compromise the integrity of ML models. Recent research has proposed defenses against such attacks [74, 66] however these still remain menacing against safety-critical ML models.

2.4.4 Targets

The generation of adversarial examples allows to attack machine learning models in different ways. As such, these attacks can be classified according to their target, in two distinct definitions.

- **Untargeted Misclassification:** the adversary is interested in having a given input classified in any class that is different from the correct one. Equation 2.3 represents this attack logic in mathematical notation.

  x' : modified input; y : class

$$M_1(x') \neq y \tag{2.3}$$

- **Source-Target Misclassification:** the adversary is interested in defining a target class in which it the model will misclassify the input to the targeted class. Equation 2.4 presents this attack where *y'* represents a specific target class.

$$M_2(x') = y' \tag{2.4}$$

2.4.5 Defenses

The growing number of new adversarial attacks being developed led to a generalized preoccupation within the community as many *real-word* machine learning applications are threatened and susceptible to such attacks. Research has been actively pursuing new defense strategies against these in order to improve the robustness of ML models.

Novel defense techniques such as *adversarial training* [37, 64] - training a model with malicious samples in order to improve its robustness - have been studied and developed to counter these type of attacks. Different defense techniques against *white-box* adversarial learning attacks have been proposed [80, 41] whereas Li et al. [40] proposed a state-of-the-art defense technique against *black-box* adversarial learning attacks.

2.5 Related Work

Since the discovery of small perturbations on inputs to machine learning models are able to manipulate its output, an observable growing research has been noted throughout the community. The power of adversarial learning in the disbelieving process of the correctness of machine learning models is becoming a more interesting and innovative topic to the research community, revealing its importance, recognition and acknowledgement.

With the growing number of fields using machine learning to operate real-world applications, the surface of adversarial learning attacks is proportionally increasing, menacing the security of such applications. In an autonomous driving scenario, machine learning has a critical role, supplying numerous models using different algorithms and techniques to the correct operation of the vehicle, such as to detect the presence of anomalies. Modern anomaly detection mechanisms rely on the use of ML models to detect the appearance of anomalies during its operation, thus adversarial learning imposes a severe threat against such models, which if failed to detect it can lead to horrid outcomes.

This section presents an overview on the existing related work regarding adversarial learning and anomaly detection systems within IoT systems, first presenting the use of machine learning in anomaly detection systems followed by works on the testing and experimentation of these models against adversarial learning techniques.

2.5.1 Machine Learning in Anomaly Detection

Anomaly detection systems encompass different mechanisms to ensure the security of systems against possible anomalies that may occur. *Rule-based* and *ML-based* are the two principal techniques used in such systems.

Rule-based approaches [25, 76], such as *traffic-based filtering* performed by firewalls, have been widely used in the development of such systems, relying on a set of threshold rules to accept or to flag/discard the input. Although its straightforward motivation, this approach struggles

when complicated and extensive rules have to be configured, and are proven to be costly, time-consuming and error prone, as is the example of current state-of-the-art DoS and DDoS attacks which are rarely detected by such approaches [62].

With the notable research interest in machine learning, a new kind of approach appeared: *ML-based* anomaly detection. This approach relies on ML models to learn patterns from and then determine if a given sample is "normal"or not [75]. The fact that ML models can learn from past behaviours and use that knowledge to understand if a given sample is anomalous without the need of human development, makes it highly desired due to low costs in maintenance and in pattern creation, specially in scenarios where there is a large amount of complex data where the creation of rules is infeasible [62].

Recent advances within this topic using deep learning techniques [45, 13] have been more present and implemented in such systems due to its performance comparing to rule-based methods.

2.5.1.1 IoT-Specifc

IoT systems have been an interesting target for research and development due to its potential and influence in modern applications. IoT, or "Internet of Things", represents a great variety of electronic devices that are connected to the Internet and are capable of communicating with each other without the need of human manipulation [46]. The growing interest in IoT systems has led to numerous built applications to help and to better integrate electronic devices so its users can more comfortably and efficiently execute their tasks. However, such systems are imminent to cyber attacks which can lead to terrific outcomes if not handled properly. With this in mind, anomaly detection has played an important role on such systems as they keep evolving and being threatened by such attacks.

Anomaly detection techniques within IoT systems are highly focused on *ML-based* approaches. Alrashdi et al. [2] focuses on smart cities cybersecurity threats thus proposing a new Anomaly Detection-IoT (AD-IoT) system based on the Random Forest Algorithm capable of effectively detecting compromised IoT devices and presenting high classification accuracy.

2.5.1.2 AV-Specifc

Regarding AVs, Wang et al. [73] proposes a deep autoencoder approach for anomaly detection in CAVs, using unsupervised learning in anomaly detection, capable of finding latent patterns of data. Theissler [67] presents an approach to detect anomalies in multivariate times series in recordings from in-vehicle networks. These recordings are captured during test drives to analyze faults. The presented goal was to detect unexpected faults without modelling effort by preventing the domain-experts to contemplate irrelevant data and rather point to the relevant portion of the internal communication recordings. Wyk et al. [72] states that the use of deep learning is a viable path to detect and identify anomalous situations. The work presents a new robust anomaly detection approach combining CNN with a well-established method, known as Kalman filtering [35], to detect and identify anomalous behaviour within CAVs.

Brake-specific anomaly detection within vehicles has been subject to different studies in order to improve the security of such safety-critical systems. Therefore, Cheifetz et al. [15] presents a detection methodology to specifically monitor the brake system of buses. Ning et al. [52] studies automobile braking anomalies in curves and Cho et al. [20] proposes a real-time anomaly detection method focused on brake-by-wire systems.

2.5.1.3 ROS-Specific

The work presented in this document uses ROS to simulate an IoT environment in conjunction with the CARLA simulator to generate and retrieve data to successfully test the built ML models with AVs-simulated data. As such, it is important to present some important related work regarding anomaly detection targeting the ROS framework.

Narayanan et al. [49] presents and proposes a new application-level, learning-based anomaly detection approach for robotic arms using the ROS middleware framework, called ROS Anomaly Detection Module (ADM) and its integration into ROS. Also, a new concept was proposed, *tolerance envelope*, to train a supervised learning model. The authors conclude that this is a promising approach although there are still modelling problems to handle. Rivera et al. [58] proposes a new approach to secure robotic systems based on ROS (*ROS-Defender*), combining a Security Event Management System (SIEM), an intrusion prevention system (IPS) and a firewall in order to provide protection against a variety of attacks. Although SIEMs, IPSs and firewall have already been used to secure computer networks, *ROS-Defender* combines them and specifically aims for robotic systems, where security is predominantly shunned. Such approach does not require any change to the existing ROS code making it a valuable asset for ROS users.

Lagraa et al. [39] presents a structured security assessment of robot cameras using ROS, with a different number of security flaws which can be exploited to take over the incoming flows from the robot cameras. An IDS is proposed to detect abnormal images coming from camera flows targeted and modified by an attacker, focusing on the vulnerable ROS camera node. The defense approach is based on images comparisons and an unsupervised anomaly detection method. Experimental results within realistic environments, such as a self-driving vehicle running on ROS, are presented demonstrating the effectiveness of the approach.

Regarding ROS 2, which may be defined as an upgraded version of ROS 1 tackling the latter's main vulnerabilities [71], Gupta et al. [31] presents some of the main issues with anomaly detection within robotic systems demonstrating the need for additional work in this specific field as well as some recommendations to address the issues covered.

2.5.2 Adversarial Learning Applications

This section presents an overview on using adversarial learning against different targets. First, a presentation of studies exploring adversarial learning against visualization-based malware detection systems, followed by IoT-specific malware detection systems. We did not find significant

research on adversarial learning within the robotic operating system (ROS), making it an interesting and innovative topic to research and contribute due to the the menacing nature of such attacks in such a widely adopted framework.

2.5.2.1 Visualization-based Malware Detection Systems

Visualization-based Malware Detection techniques try to detect different features in executable files by converting them into bitmap images. More specifically, they convert malware binaries into gray-scale bitmap images so that the different binary files can be visually analysed and classified by malware researchers, by observing significant visual similarities in malware files from the same family [32, 50].

Khormali et al. [36] proposes a new approach (COPYCAT) to generate adversarial examples specifically designed for visualization-based malware detection systems. This paper explores the generation of adversarial examples with five different algorithms in order to maximize the attack success rate while still maintaining the functionality and executability of the original files. A practical approach was taken and attacks were experimented and its results were analysed. This study successfully concludes that deep learning models are still vulnerable to adversarial examples, more precisely in deep learning malware detection, and that COPYCAT is able to achieve high attack success rate. Another approach is presented by Liu et al. [42] where a new attack to these types of malware detection systems is proposed, named the Adversarial Texture Malware Perturbation Attack (ATMPA). The tested ML-based detectors were all unable to detect the malware correctly, proving to be a menacing and effective approach to attack such mechanisms.

2.5.2.2 IoT-specific Malware Detection Systems

Abusnaina et al. [1] studies how IoT malware detection techniques using control flow graph (CGF)-based features and deep learning networks (DNNs) are vulnerable against adversarial learning, presenting two different approaches to create adversarial examples. It concludes that there is the need for more robust IoT malware detection tools against this threatening technique. The authors clearly state that the impact of adversarial learning against deep learning-based IoT malware detection systems is subject to little research and that such implication may lead to disastrous outcomes in practice. Morgulis et al. [48] focuses on the security of self-driving vehicles, presenting a process to generate *adversarial examples* (in the form of traffic signs) capable of fooling a wide range of classifiers. Sagduyu et al. [61] presents developed adversarial learning techniques to both attack and defend IoT systems considering a network using an IoT transmitter.

2.5.2.3 AV-specific

Regarding the study of adversarial learning in AVs Cao et al. [11] presents an extensive study on adversarial learning attacks against LiDAR-based perception in AVs whereas Sharma et al. [65] studies such attacks against modern machine learning and deep learning models, presenting a novel approach to generate adversarial samples through the use of an ML model. Qayyum et

al. [57] presents an exploratory analysis on the challenges posed by adversarial learning attacks to CAVs, describing the multiple type of attacks that can be launched against the various ML components of the system as well as a conducted survey of the attacks and defenses of adversarial learning in current literature.

2.5.3 ROS Security Frameworks

With regard to frameworks targeting ROS security, Rivera et al. [59] presents *ROSploit*, a new exploitation tool for simulating attacks against ROS systems. A similar tool was presented by Dieber et al. [21], named *ROSPenTo*, which is defined as a tool for semi-automated penetration testing. Although not adversarial learning-specific, it is important to note the existence of both tools as they aim to help researchers and developers to test their robotic systems against different attacks.

2.6 Summary

Machine learning-based anomaly detection has been subject of research due to its remarkable capability of learning how to efficiently learn from past behaviors and correctly predict future anomalous behaviors. Numerous approaches were presented focused on both IoT and AV systems, including brake-specific in autonomous vehicles as well as ROS-specific.

An increasing interest of adversarial learning has sourced different attacks and defenses, permitting their application in different scenarios studied by the research community. The lack of non-existing studies on the application of adversarial learning techniques against brake-specific machine learning-based anomaly detection of autonomous vehicles make it an innovative and important topic to research, contributing to the security and development of these intelligent systems.

Chapter 3

Carla-ROS Bridge Anomaly Detection Component

The construction of a machine learning model for detecting brake anomalies within simulated vehicles using the Carla simulator, focuses on performing a group of different techniques which aim to improve the model's capability of successfully learning and consequently predicting both normal and anomalous braking situations. However, the existence of a model alone is not sufficient for detecting such anomalies in real-time as this is presented as an external entity to the simulator, which has no access to the information transferred between its components. Therefore, the first phase of the development of this work consisted on defining a well-designed and robust infrastructure represented by a novelty ROS package, fully-integrated with Carla-ROS bridge, named *carla_anomaly_detection* and commonly referred to as the anomaly detection component. Such component is capable of integrating the brake-specific anomaly detection model with the ongoing simulation, so that the vehicle's data is accessible to be used for prediction and understand if the vehicle is or not in an anomalous state. The creation of this component was essential to the subsequent work phases as it was used for exposing the different created machine learning models to the ongoing simulation and thus conduct an extensive testing phase based on their performance, later presented in chapter 4. This chapter presents the infrastructure of this new component and how it operates internally.

3.1 System Integration

As presented in chapter 2, Carla-ROS bridge provides a communication channel between ROS components and the Carla simulation itself, allowing external interactions, such as controlling the vehicle or retrieving real-time information from the multiple sensors attached to it. Having such tool to retrieve real-time information of an ongoing simulation served as a basis for the implementation of a new external ROS component focusing on detecting anomalies of the current simulated vehicle.

To ensure the new component's software development process was correctly conducted, a set of requirements were defined forehand consisting in the following:

1. Read real-time information generated by the vehicle's sensors.
2. Parse and filter the information read depending on the machine learning model features.
3. Construct an internal workflow for acting upon the vehicle in case an anomaly is detected.

With regards to such requirements, the proposed software solution was to create a new ROS package that makes use of Carla-ROS bridge capabilities and integrates the new anomaly detection entity into the system. The ROS component was designed so that it is completely integrated with Carla-ROS bridge hopefully making it a useful tool for the research community that study anomaly detection in autonomous vehicles.

3.2 Architecture of the Component

The created component is composed by two ROS nodes with different goals yet maintaining a one-to-one relation between them. These nodes were designed with both flexibility and scalability in mind, following best software design practices in order to deliver a robust component. Figure 3.1 presents a diagram representing these nodes.

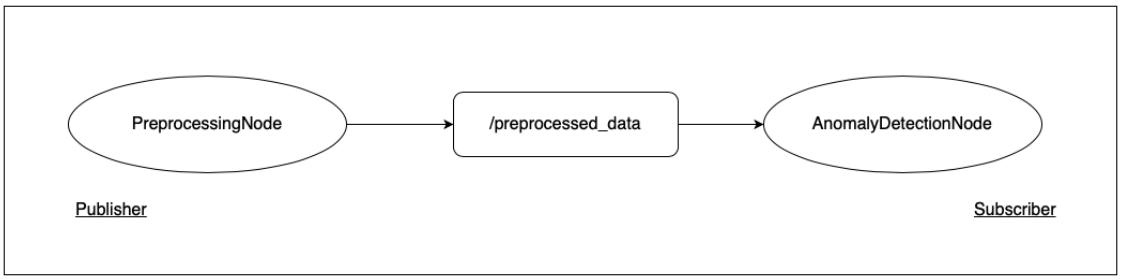


Figure 3.1: Diagram representing the entities and their interaction of the created anomaly detection component.

3.2.1 Preprocessing Node

Processing of the raw published information flowing through the different Carla-ROS bridge topics is essential so that the correct features are filtered and parsed before feeding them to the model for later prediction. This ROS node focuses on subscribing to the topics containing the features that the model is expecting and aggregating its values for later publishing to a topic (/preprocessed_data) exclusively listened by the anomaly detector node. Table 3.1 presents the topics to which this node publishes/subscribes for its successful operation.

Table 3.1: Preprocessing Node topics interactions.

Topic	Role	Description
/preprocessed_data	Publisher	Sample to be predicted.
/carla/ego_vehicle/vehicle_status	Subscriber	Ego vehicle's sensors information.
/carla/ego_vehicle/odometry	Subscriber	Ego vehicle's odometry information.

3.2.2 Preprocessed Data Topic

The passing of information between ROS publisher and subscriber nodes is dependent of a entity known as topic. To allow communication between the two nodes of this component, a new topic was created in order to support the transfer of data from the preprocessing node to the one responsible for feeding the received information as input to the anomaly detection model.

This topic supports a specific type of data that is only expected to be understood by the entities described here as it is intended to transfer messages of non built-in ROS messaging types. Because of this, a new ROS message type was created, named `SampleArray`, for defining the type of information transferred through this topic. This messaging type represents a sample (array of values) to be received by the subscriber node that is responsible to process and consequently provide it as input to the anomaly detection model for prediction. Therefore, one can abstractly define the information flowing through this topic as real-time samples to be predicted by the anomaly detection model. Listing 3.1 presents the underlying data structure for the created messaging type.

```
1  float32[] data
```

Listing 3.1: Contents of the created ROS message type for communication between the anomaly detection component nodes: SampleArray.msg

Although this messaging type has the same underlying structure as the one defined in the already existing `rospy_numpy` message type, defined in the ROS python client library [69], it offers the possibility of altering it to any other type of data structure(s) that may be required. For instance, if a different anomaly detection model is used and this does not support `float32` as input but `String` instead, the `SamplesArray` messaging type could be easily changed as opposite to the `rospy_numpy` messaging type which is a part of the ROS client library and thus should not be modified. Another advantage of having a custom messaging type is that additional restrictions may be implied to it, as for the current context one could have defined the maximum number of elements of the `float32` array of each message, avoiding potential data overflows. Such feature improves both the security, usability and customization of the new package.

Listing 3.2 presents the import command to start using this new messaging type inside the package's nodes.

```
1  from carla_anomaly_detection.msg import SampleArray
```

Listing 3.2: Import command of the created `SampleArray` message type.

Table 3.2: Preprocessing data topic information.

Name	Message Type	Description
/preprocessed_data	`SampleArray`	Sample combining the feature values for prediction.

3.2.3 Anomaly Detector Node

Following the preprocessing of the relevant information that the model is expecting, conducted by the preprocessing node, and after arrival of the samples for prediction through the `/preprocessed_data` topic, the anomaly detector node is ready to start its computation. As such, this node first subscribes to this topic for the sample retrieval process, which for every message received it starts its internal processing workflow, represented in figure 3.2. Due to the requirements initially defined, this component has a privileged level of access to the control of the vehicle in order to act upon it whenever an anomaly is detected. As such, this node is also a publisher of some of the available Carla-ROS bridge topics that allow the dispatch of vehicle control messages.

3.2.3.1 Anomaly Count Threshold

An anomaly may be defined as an observation which does not satisfies "normal" values constraints. Because a vehicle is such a complex system containing numerous sensors, each communicating with different entities, there may be rare transient events that may cause unintended corruption of values (i.e. corruption of data from a sensor message) which do not represent a real anomaly. Therefore, to handle such edge cases a simple yet robust mechanism is proposed consisting on defining a "safe-count" limit, also known as *threshold*. This mechanism is defined as the margin of contiguous anomalous samples predicted before assuming that there is indeed an anomaly present in the vehicle's operation.

This mechanism was implemented recurring to a simple counter which is incremented each time an anomalous sample is predicted. Whenever the threshold limit is crossed then an anomaly is infered and the component is free to act upon the vehicle. Note that this counter is reset every time a sample is predicted as non anomalous, thus permitting only the count of contiguous anomalous samples to cross the defined threshold. To define its value, different factors were taken into account, such as the fixed time-step that samples were generated from the simulation (0.05s) as well as the underlying risk of losing ROS messages during transit. Therefore, during the development of this work, the threshold value was set to ten (10) samples as it assumed to be a reasonable value for differentiating an anomaly from a transient issue.

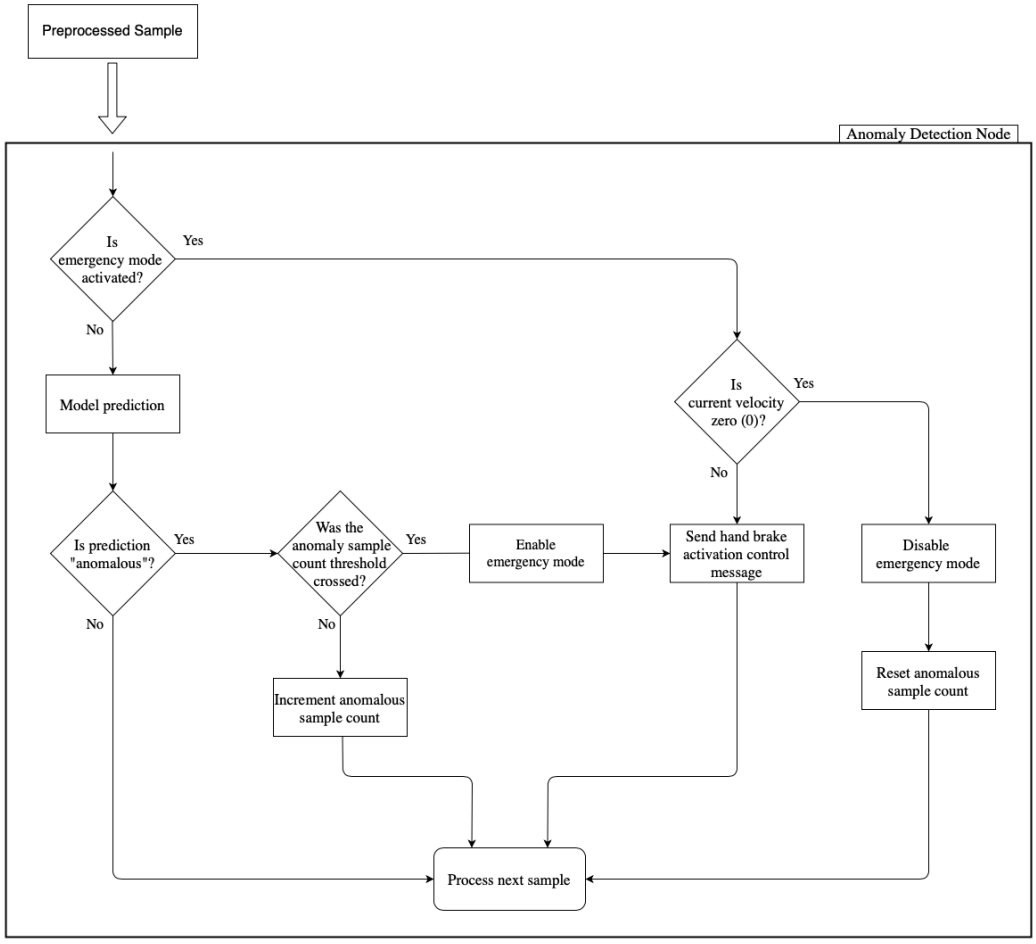


Figure 3.2: Fluxogram representing the workflow of the anomaly detection ROS node.

3.2.3.2 Control Action

The control action represents the action to take whenever an anomaly is detected. This component has a privileged level of access to the vehicle, allowing it to send its own control messages. Because this work focuses on creating a brake-specific anomaly detector model, it is reasonable to assume that the component's action has to be related with the correction or elimination of non-braking behaviors. Therefore, the action to take whenever an anomaly is detected is to publish an emergency brake control - activation of the hand brake - to the `/carla/ego_vehicle/vehicle_control_cmd_manual` topic, responsible for propagating manual controls to the simulated vehicle. Listing 3.3 presents the configuration of the emergency control message.

```
1  control_msg = CarlaEgoVehicleControl()
```

```
2   control_msg.throttle = 0
3   control_msg.hand_brake = True
```

Listing 3.3: Vehicle control message sent whenever the emergency mode is activated.

The defined behavior is not intended to be directly applicable in real-world scenarios because of the security of the world that surrounds it and therefore it is only presented regarding the context of this work. Such behavior may lead to danger situations where such an abrupt brake may cause other vehicles to crash against it depending on the velocity and distance relative to the vehicle they are at.

3.2.3.3 Model Predictions Topic

The predictions of the model allow the understanding of the its performance when used in a simulation scenarios. Therefore, in order to extract and expose this information to external components, a new topic was created to which this node publishes the model's predictions in real-time. The topic, named `/ad_predictions`, was an important tool during the development of this work as its information was recorded together with the samples to be predicted, to later analyse and calculate metrics based on the number of correct predictions by the anomaly detection model. Table 3.3 presents a detailed description of this topic.

Table 3.3: ad_predictions topic description.

Name	Message Type	Description
/ad_predictions	Int32	Exposes the model predictions for the incoming samples.

This topic may also be useful to other nodes which want to obtain the predictions of the model in real-time as well as for future *debugging* purposes.

3.2.3.4 Emergency Mode Alert Topic

Whenever an anomaly is assumed to be happening with the vehicle's brakes, the emergency mode of this node is activated, resulting in a hand brake activation control command message being sent to the vehicle, as earlier described in section 3.2.3.2. However, for such control messages to be successfully transmitted to the simulator and consequently applied to the simulated vehicle, it is required that the node first publishes a true value (boolean) to the `/carla/ego_vehicle/vehicle_control_manual_override` topic in order to override the vehicle control commands published in the alternate topic `/carla/ego_vehicle/vehicle_control_cmd`, commonly used by the autopilot mode. The manual control feature present in the Carla simulator allows the input of keyboard controls, later transformed to vehicle controls commands through the `/carla/ego_vehicle/vehicle_control_cmd_manual` topic which is the same used by this node to send the emergency brake control commands. Therefore, a conflict arises when both the emergency mode and manual control are activated causing a mix of information published to the topic, possibly causing unexpected behavior in the vehicle's operation. To solve this issue, an

additional topic to connect both this and the `carla_manual_control_ego_vehicle` node was created in order to propagate the emergency mode and thus disable the commands sent by the `carla_manual_control_ego_vehicle node`, as represented in figure 3.3. Table 3.4 presents this topic description and figure 3.3 describes how both nodes communicate to synchronise which node is allowed to send a control message to the vehicle.

Note that when the autopilot mode is enabled this won't happen as the topic to which its generated controls are published is different (`/carla/ego_vehicle/vehicle_control_cmd`).

Table 3.4: Emergency mode state topic description.

Name	Message Type	Description
/anomaly_detection/emergency_mode	`Bool`	Emergency mode state.

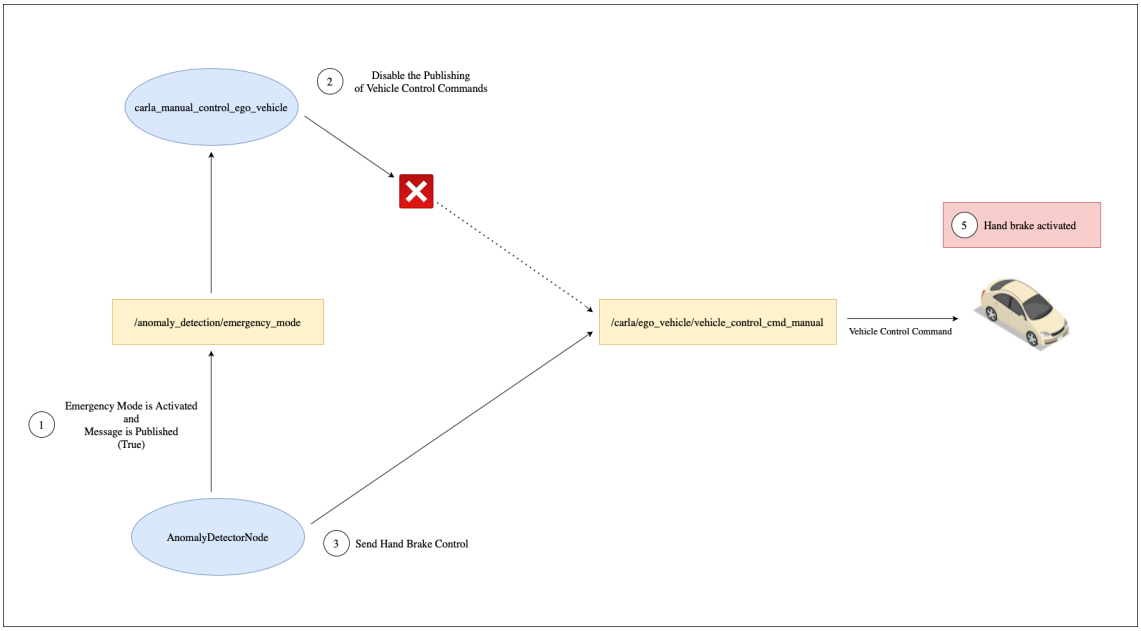


Figure 3.3: Diagram representing the entities and their interaction of the created anomaly detection component.

Table 3.5 presents the set of topics that this node interacts with as well as its role in them.

Table 3.5: Anomaly Detector Node topics interactions.

Topic	Role	Description
/preprocessed_data	Subscriber	Sample combining the feature values for prediction.
/carla/ego_vehicle/vehicle_control_cmd_manual	Publisher	Vehicle control commands.
/carla/ego_vehicle/vehicle_control_manual_override	Publisher	Override of vehicle control commands.
/ad_predictions	Publisher	Model predictions.
/anomaly_detection/emergency_mode	Publisher	Emergency mode state.

At last, figure 3.4 presents a diagram of a simulation run using the anomaly detection component.

3.3 Summary

The presented novel anomaly detection component permits the simple integration of machine learning-based anomaly detection models in Carla simulations making use of Carla-ROS bridge. With it, the conduct of simulations using such models is now possible and thus permits an intuitive and ready-to-use tool for research of anomaly detection in autonomous vehicles under realistic conditions.

Following the creation of such component, the development of a brake-specific anomaly detection model was commenced using this component as the infrastructure for its use in autonomous driving simulations and consequently evaluate its performance in such situations.

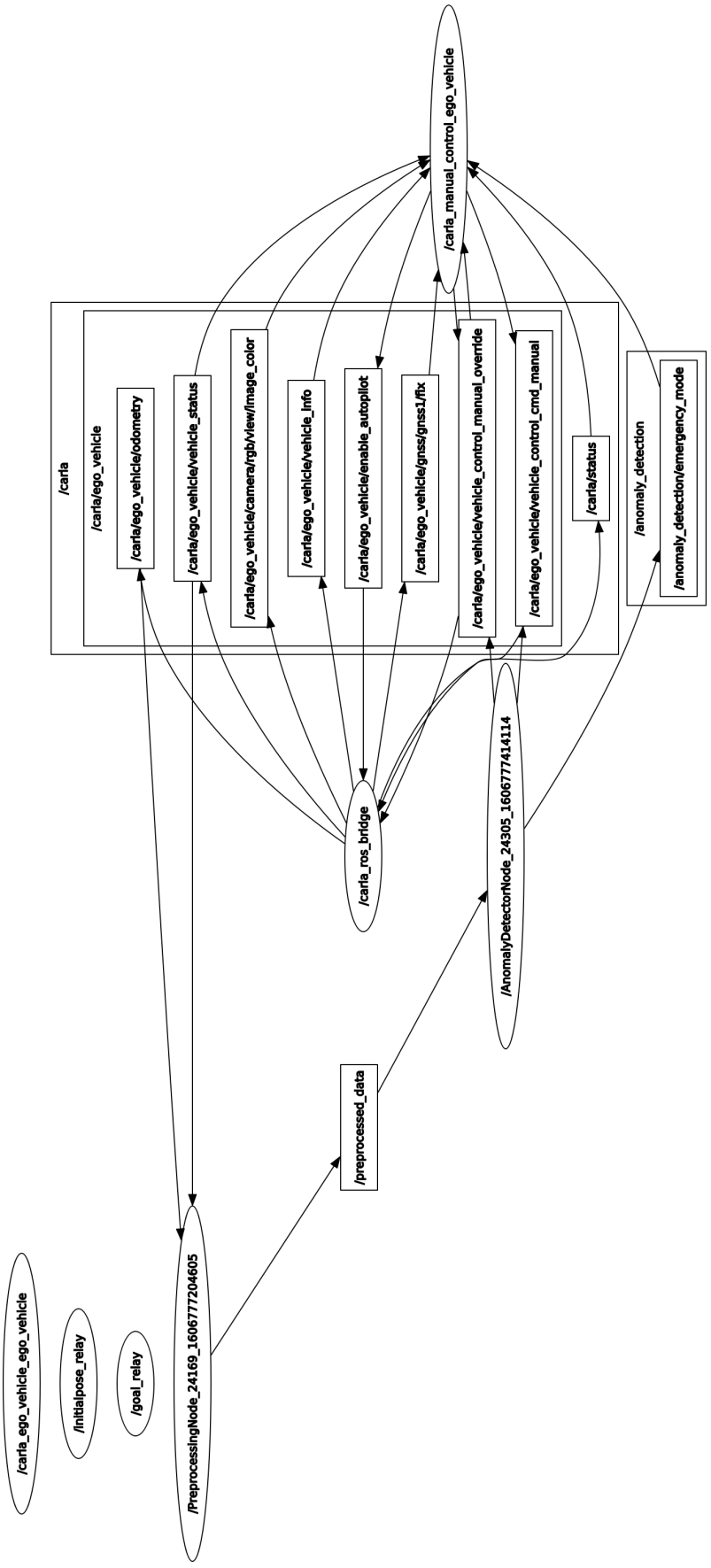


Figure 3.4: Anomaly detection component integration with Carla-ROS bridge.

Chapter 4

Brake-Specific Anomaly Detection Model

Anomaly detection has proven to be a key subject when implementing the security of any information system. Such process provides extra security capabilities to a system's operation, offering understanding on whether a current situation is considered to be normal or anomalous. Machine learning has been a commonly adopted paradigm when implementing anomaly detection features due to its ability to efficiently learn from past behaviors and accurately detect unprecedented ones.

In this chapter, we present the process of creating a machine learning model for classification of different possible braking situations within an autonomous vehicle. The methodology of the work is first presented, explaining details on how different models were created and tested against different scenarios in order to have significant information for conducting a relevant comparative analysis and hence select the most appropriate model for the problem. Subsequently, the process of extracting and collecting relevant data from the Carla simulator for input to the machine learning models is introduced as well as the different set of data preprocessing techniques applied when creating the different models. Distinct generalized approaches are presented and discussed, using both supervised and unsupervised learning paradigms, each explaining and presenting the metrics of its performance and efficiency when tested against previously unseen samples, introducing an additional testing concept referred throughout this document as *production testing*. At last, the in-depth comparative analysis of the models is presented as well as the selected model for brake-specific anomaly detection.

4.1 Methodology

To create a robust and accurate anomaly detection component, three different machine learning approaches were studied: one-class, binary and multi-class classification. Implementing and testing such approaches together with a detailed comparative analysis of each results reveals not only the most appropriate machine learning model from the studied set, but also important characteristics

of the data and how the different models adapt to and learn it. Figure 4.1 presents the methodology used for the creation and selection of the anomaly detection machine learning model.

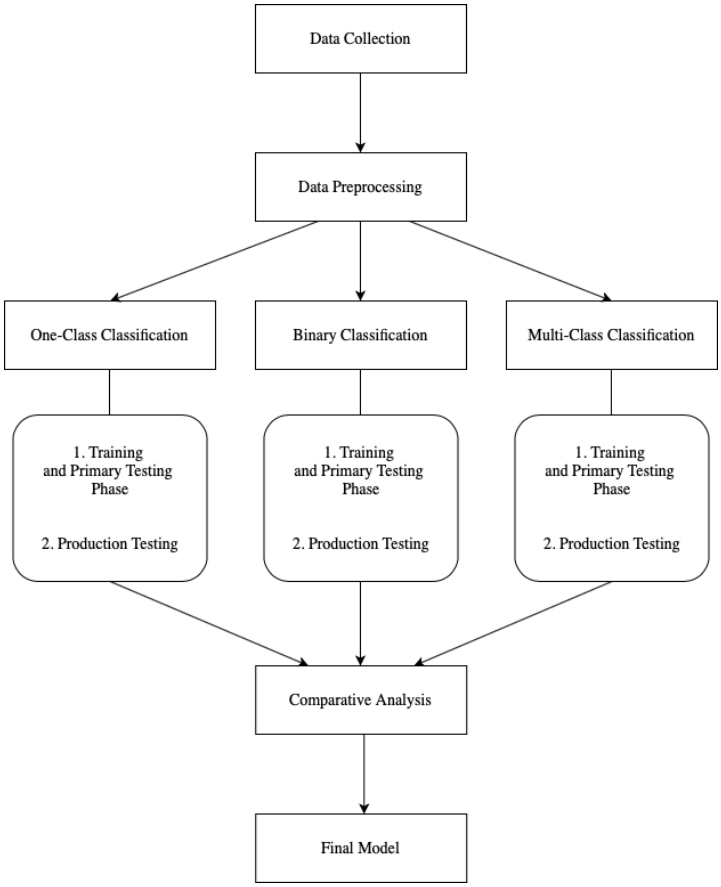


Figure 4.1: Methodology followed for implementing, testing and comparing the different machine learning models.

For each of the approaches presented both a training and a production testing phase was conducted. By doing so, there is a higher level of confidence on the model's behavior when presented to real-world data, thereby delivering more relevant information for later comparison and consequently the decision of the most appropriate model.

4.1.1 Metrics

Metrics are important as they present a simple representation of a model's performance. To better compare how the different models performed in the different situations they were submitted to, different metrics are presented throughout this document. To contextualize the reader, this section describes the formulas of these. Table 4.1 presents a confusion matrix which exposes such entities and how they are related with each other. This is an extremely useful tool for estimating a model's performance upon classification problems.

Table 4.1: Confusion matrix.

		Prediction	
		Positive	Negative
Label	Positive	TP	FN
	Negative	FP	TN

Precision

Measures the portion of the predicted positive values that are true positives. This metric is also known as Positive Predicted Value (PPV) and it is defined as:

$$Precision = \frac{TP}{TP+FP} \tag{4.1}$$

Recall

Measures the portion of the true positives that were correctly identified. This metric is also known as True Positive Rate (TPR) and it is defined as:

$$Recall = \frac{TP}{TP+FN} \tag{4.2}$$

F_1-Score

Represents the harmonic mean of the precision and recall.

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4.3}$$

Accuracy

Measures the portion of correct predictions.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{4.4}$$

Note that all the metric values presented throughout this work were rounded up to two decimal places.

4.1.2 The Training and Primary Testing Phase

Machine learning algorithms rely on information to automatically learn its characteristics and correctly predict previously unseen data. Their learning process is characterized by an iterative procedure, known as *fitting*, which focuses on determining a satisfying mapping function which correctly maps each given input to its expected output. This process is also known as training the model as it will determine how it will behave when predicting inputs not present in its training data. In order to efficiently evaluate a machine learning model's accuracy it is crucial that the

data used to test it is not the same as the one given as part of its learning (training) process as it will most likely present satisfying results which do not reflect its realistic performance against unseen data. Therefore, a common practice to adopt when working with machine learning is to partition the data in different sets, known as training and test sets, in which only the training data set will be used for the model's learning process as opposite to the test set which will be used to evaluate its performance against unseen data and thus presenting relevant results of the model's performance against it. Such practice offers important benefits such as to understand if the model is suffering from well-known machine learning problems known as *overfitting* (occurring when the model learns the training data too well and fails to generalize to unseen data) or *underfitting* (when the model does not learn the training data causing it to poorly perform in the context of the problem).

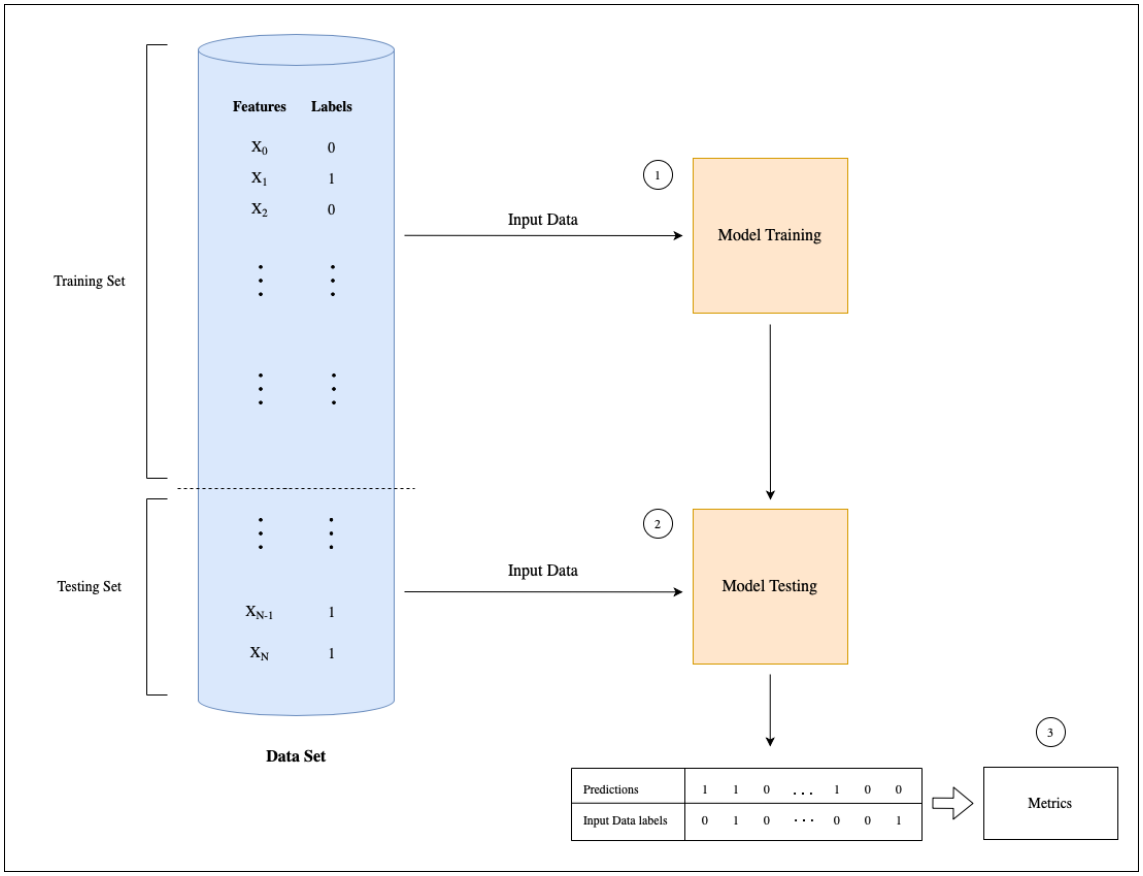


Figure 4.2: Training and primary testing phase representation.

There is an additional concept usually adopted when working with deep learning solutions, known as validation, as these have a high probability of *overfitting* to the training data. Such concept consists on partitioning the original data set into three individual data sets (training, validation and test set) which the model will use to learn on the training set and for each iteration of the learning process, will test against different, unseen data from the validation set as opposite to its own training data as presented in the previous approach. This allows the model to evaluate its

performance on each iteration and, consequently, fine-tune its parameters. This concept was used in both binary and multi-class classification approaches as these used deep neural networks for targeting the current problem.

Hyperparameters are variables which define a model's learning process, highly influencing the model's ability to successfully learn the training data peculiarities and correctly predict on unseen samples. Throughout this work, a rigorous process known as manual hyperparameter tuning was conducted for each created machine learning model. Such process is defined by the definition of different hyperparameter values (i.e. number of epochs or learning rate) before its training process is commenced in order to optimize the search for the model with the highest performance. It is intuitive that there is an infinite space of possible hyperparameters tuning combinations, being this a common research problem which current literature [6, 77] explores by presenting different algorithms whose primary goal is to find optimal combinations. Although such algorithms could have been used, manual tuning was the elected approach as it offers important benefits such as a better understanding of the collected data and the relevance of the different hyprparameters weights on the model's performance.

After a thorough comparison of the metrics obtained for the different combinations, the one which presents the most satisfying results is selected and formerly submitted to the additional testing phase, production testing.

4.1.3 The Production Testing Phase

Although the analysis of the initial metrics is critical when choosing the most appropriate model for the context of the problem, such metrics may not be sufficient to understand the full extent of the model's capabilities. Because the ultimate goal of each created model is to detect whether a brake anomaly is present or not, it was decided to implement an additional testing phase, due to the infinite possible driving scenarios that may occur during the operation of an autonomous vehicle. Note that in the previous testing phase, primary testing, a portion of the collected data is selected for testing the model against previously unseen data, however such data will tendencially have similar values of the ones in the training data set as these splitted data sets will correspond to data collected from the same simulations. Therefore, it is ambiguous to only rely on such testing as the model's performance against it might not represent its authentic behavior when submitted against different simulations. With the use of production testing, a new dimension of the evaluation of the model's performance is introduced and thus contributing to a higher level of confidence when deciding which model to choose.

At its core, production testing focuses on submitting a model into a live driving simulation to have the model predicting previously unseen samples generated in real-time, therefore simulating its behavior in a real-world scenario. Figure 4.3 presents a visual representation of the workflow executed for each simulation of this phase. Note that this type of testing requires the integration of the models to test in the simulation's ecosystem, thus, the created anomaly detection component presented in chapter 3 was used to expose the models to the real-time samples in order to classify them. Also, to have significant data for later comparison of each model's performance, this phase

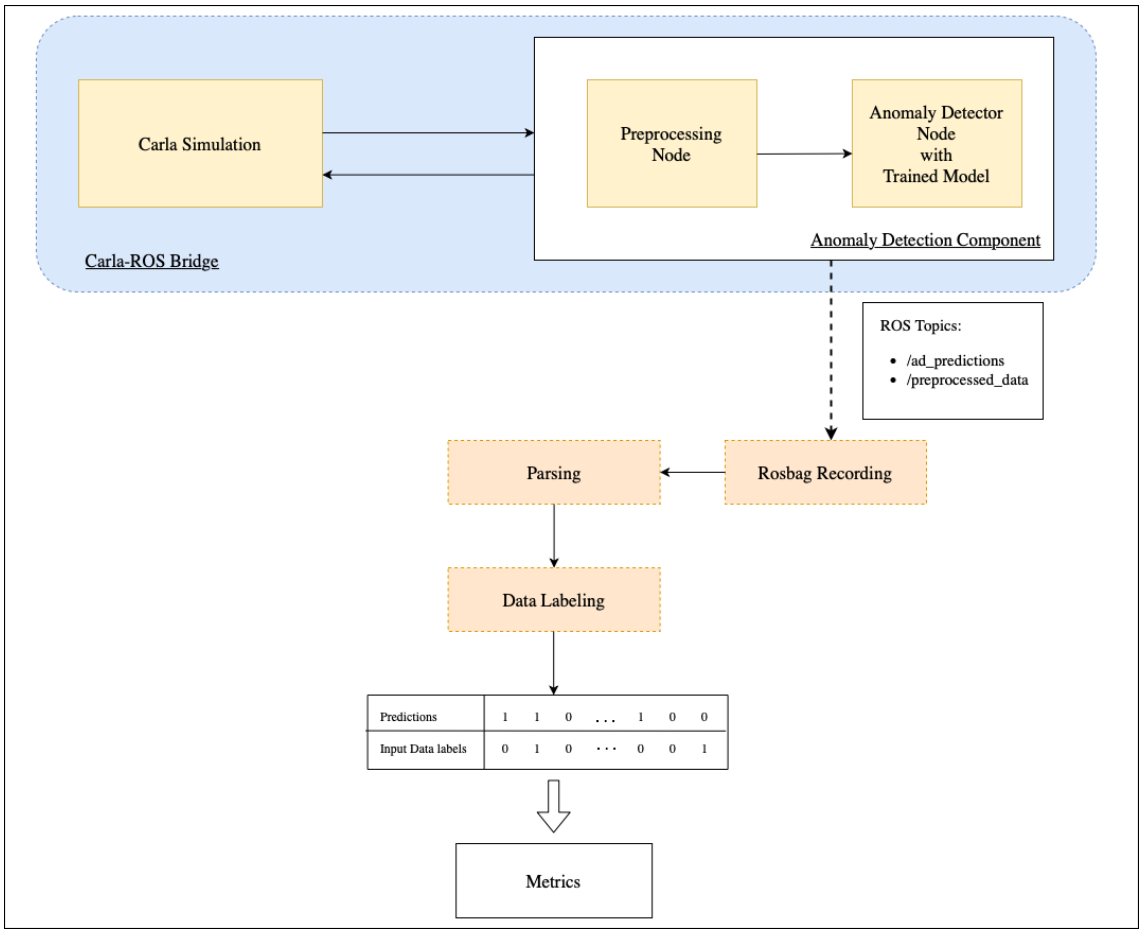


Figure 4.3: Workflow executed for each production testing simulation.

consisted on five different simulation runs each involving distinct brakes failure combinations, as presented in table 4.2, in order to analyse how the model behaves to previously unseen data but also regarding the level of severity of the anomaly.

Table 4.2: Description of the brakes failing for each production testing simulation.

Simulation	Brake(s) failing
#1	-
#2	FR
#3	FR, RR
#4	FR, RL, RR
#5	FL, FR, RL, RR

Note that throughout this work common abbreviations are used for referring to production testing simulation runs as well as to each of the vehicles brakes. The former are represented by the "#" symbol followed by a number (i.e. #1) and refer to the descriptions presented in table 4.2. To contextualize the reader to the location of the brakes and its respective abbreviations, figure 4.4 presents its visual representation.

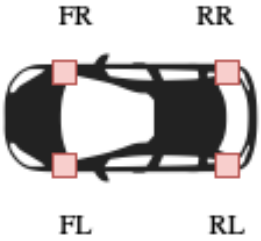


Figure 4.4: Location of the brakes in the simulated vehicle.

To obtain each simulation metrics, a supplementary processing stage had to be performed. Because this testing phase consists on submitting the model to a live simulation environment, there was the need of mapping each sample generated to its prediction. This information was not directly available due to the system's design. Therefore, the ROS topics responsible for transferring samples for prediction (`preprocessed_data`) and for exposing their values (`ad_predictions`) were recorded using the rosbag tool, running the command presented in listing 4.1.

```
1  $ rosbag record /preprocessed_data /ad_predictions
```

Listing 4.1: Rosbag command to record data used for production testing analysis.

Depending on the simulation, the generated samples were labeled as either normal (#1) or anomalous (#2-#5), permitting to later compare each label with its prediction and obtain the different metrics for each production testing simulation run. The metrics collected from the execution of this testing phase will inevitably contribute to a more conscious and confident decision on which of the created models suits best for the current problem: detecting a brake anomaly.

4.2 Data Collection

The gathering of information plays an essential role when it comes to the application of machine learning techniques. Collected data will be fed as input to the ML models which will internally derive a function responsible for mapping an input to some output value. Therefore, it is essential that the data fed to the model is as relevant and clear as possible so that it is able to fully learn from it and later correctly map unseen inputs to the correct output value.

The development of a brake-specific anomaly detection model requires a data set composed of a high volume of both normal and anomalous samples so that, as stated before, the model can successfully learn how to distinguish samples from each class and thus guarantee future accurate predictions. To retrieve information generated by the simulator, the Carla-ROS bridge software framework was used together with the built-in ROS's rosbag tool permitting to record messages published in relevant topics. However, because the target goal was to detect braking anomalies

within an autonomous vehicle, not all topics contained relevant information for the models to learn how to detect a brake anomaly. In fact, the vast majority of information published to the different Carla-ROS bridge topics revealed an absence of relevance for the problem in question, either by carrying non-vehicle data (i.e. world information) or non-relevant sensor data for the detection of anomalies in the vehicle's braking system (i.e. LiDAR images). Therefore, a careful analysis on each topics information was conducted disclosing two topics, presented in table 4.3, as the ones containing useful information for detecting brake anomalies.

Table 4.3: Topics selected for data collection.

Topic	Description
/carla/ego_vehicle/vehicle_status	Ego vehicle's sensors information.
/carla/ego_vehicle/odometry	Ego vehicle's odometry information.

It is important to note that, as presented in chapter 2, Carla allows both synchronous and asynchronous simulation runs depending on both the user's goal and platform requirements. To effectively collect meaningful and realistic information, synchronization was essential as it groups the different sensors information at a fixed time-step, in order to ensure that the data published at a given timestamp corresponds to the data published by all sensors at approximately the same time. Following the topics filtering process, the recording of multiple simulations data began thus creating the training data set for for input to the machine learning models.

A brake anomaly is something that may be intuitive to human beings, because of its impact and noticeable effect that is perceived when driving a vehicle. However, machine learning techniques aren't quite as intuitive. Such techniques require the understanding of these effects by inspecting the data generated by the vehicle's sensors in real-time. It is easily understood that there is an infinite universe of combinations between the different values of the generated data by each of the vehicle's sensors (i.e. velocity, acceleration, odometry, among others). Therefore, the data collection process is responsible for aggregating the highest number of possible relevant scenarios, in this case brake-focused scenarios, so that previously unseen data samples can be correctly classified and the model's efficiency is at its best. To address this issue and minimize the risk of failing to successfully predict such samples, multiple simulation scenarios were covered so that the model had relevant information to learn the different characteristics of both normal and anomalous brakes. Table 4.4 presents the different scenarios from which driving data was gathered, each representing a different brake combination as well as their number of samples. Note that these simulations were conducted using both the Carla simulator autopilot mode, which has a fixed speed limit, as well as manual controls, in order to improve the model's robustness against wreckless behaviors.

In summary, the data collection process was carried by recording the information published in the valuable ROS topics, presented in table 4.3, per each brake combination, presented in table 4.4. By doing so, a rich and diversified data set was collected so that the to-be created anomaly detection model was robust and accurate when detecting different possible anomalous braking

Table 4.4: Description of the scenarios used for data collection.

Brakes Availability				# Braking Samples	# Total Samples	Driving Mode *
FL	FR	RL	RR			
✓	✓	✓	✓	64139	188546	A,M
✓	✓	✓	×	794	2957	A,M
✓	✓	×	✓	2993	9640	A,M
✓	✓	×	×	1379	5286	A,M
✓	×	✓	✓	1260	6342	A,M
✓	×	✓	×	2213	7405	A,M
✓	×	×	✓	1960	6028	A,M
✓	×	×	×	1853	6096	A,M
×	✓	✓	✓	1137	4888	A,M
×	✓	✓	×	1306	5640	A,M
×	✓	×	✓	2977	10000	A,M
×	✓	×	×	1826	6414	A,M
×	×	✓	✓	2341	7530	A,M
×	×	✓	×	4062	9920	A,M
×	×	×	✓	2732	9001	A,M
×	×	×	×	4313	13544	A,M

*A = Autopilot; M = Manual Control

situations. Figure 4.5 presents the diagram of the state of the ROS computational graph configuration upon recording the relevant information for the data set. After the recording process was finished, multiple rosbags files (`.bag`) were present, which later required a parsing process in order to convert these to common formats for later analysis and application to the machine learning models.

4.3 Data Preprocessing

The data collection process presented in the previous section represents a crucial portion of the creation of a machine learning model as the gathered data will serve as the basis for the model's learning process and consequently influence its capability of correctly predicting new, unseen samples. However, in most of the cases, it is not enough to simply collect and feed the raw, untreated data to a machine learning model. Such data may contain numerous irregularities that need to be addressed in order to improve the model's performance or even to be able to start consuming it for the training process. Because of this, a procedure was conducted for each of the models created, focusing on the processing of the data before feeding it as input. Such procedure is known as data preprocessing and may encompass a different number of techniques such as data cleaning, normalization and data transformation, which in most cases result in a significant positive impact on the performance of machine learning models.

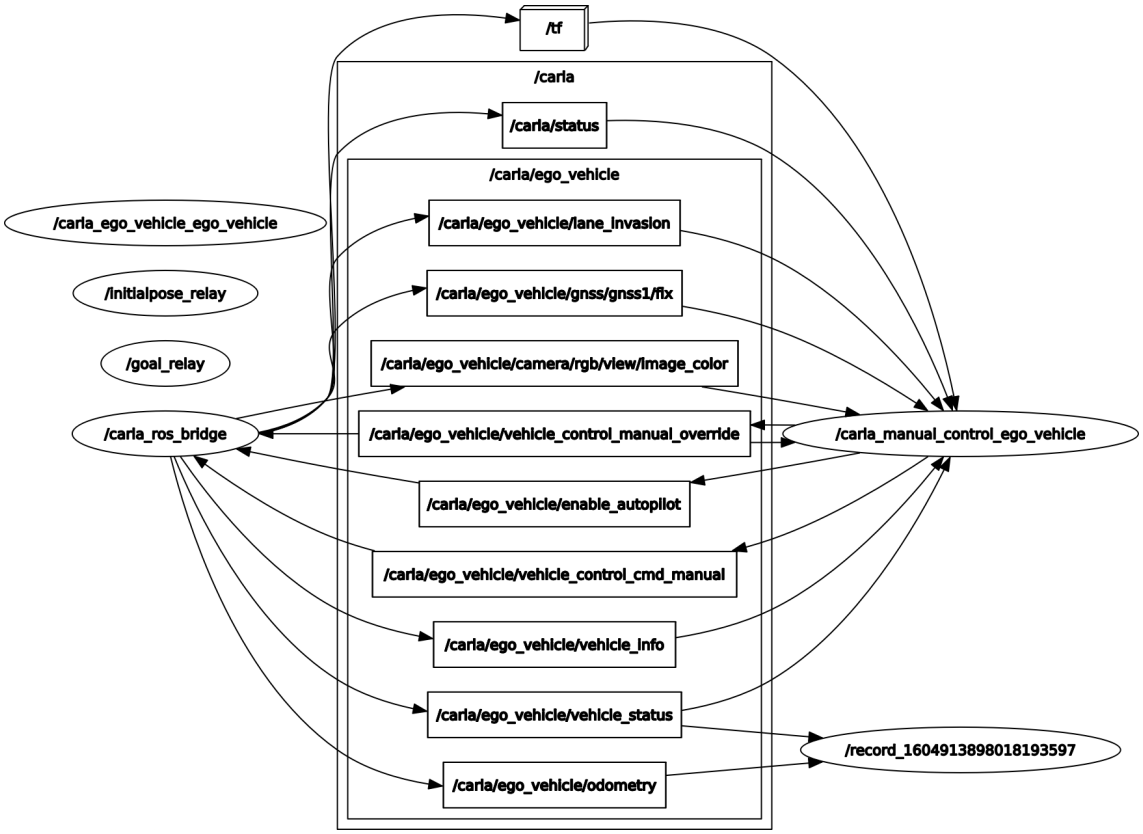


Figure 4.5: Diagram representing the active topics upon data collection.

4.3.1 Parsing

As part of the preprocessing techniques applied in the development of this work, parsing was essential as it permitted to convert the .bag files containing the collected data into both a human-readable and supported format, .csv, for data analysis tools. Since .bag files are specific to ROS applications and the lack of an official tool for converting such files into common text formats, a script, presented in C.1, was created to do so. Figure 4.6 presents a visual representation of the script's operation.

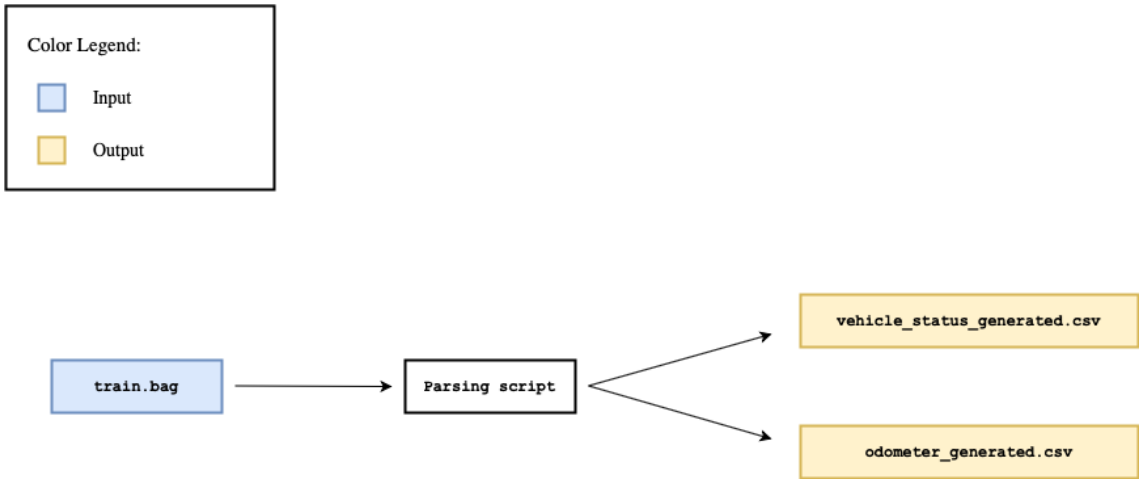


Figure 4.6: Diagram representing the behavior of the script for parsing rosbag (.bag) files.

As it can be seen, the script creates a .csv file per topic. Although this is extremely useful as it decouples the different topics information, the integration of the data spread across the script-generated files was required in order to have the information of a simulation in a single file. However, such process is not trivial as synchronisation must be guaranteed so that each row of the new file corresponds to the information published at the same moment. To achieve this, ROS messages header information was used, as these contain metadata about each message published time, resulting on the messages published in the different topics at the same moment to be merged and thus composing a sample.

After applying this script to all the collected .bag files, there was the need to integrate all the samples in a single file and thus define the whole training data set which would then be fed to the models. However, before doing so there was the need of labeling each sample.

4.3.2 Data Labeling

Labeling data is an important step of data preprocessing as it allows to categorize each of the collected samples into a specific class, regarding an arbitrary condition. Within supervised learning, this is the guidance that models seek for learning the relation between the existing features and its labels and thus understand the properties of each target class for later prediction.

In this work, data labeling was executed after the parsing phase was finished. After its operation, multiple single files containing each simulation samples were created and therefore it was required to first label each sample as either normal or anomalous before gathering all of them together and form the entire collected data set. As such, depending on the situation to be labeled, for each sample, a condition was applied. Because these files were known to correspond to some braking behavior (either normal or anomalous), all the brake messages that were presented in the data set were categorized as either normal, for normal braking situations (corresponding to the first row of table 4.4), or as anomalous for situations where at least one of the brakes was failing (corresponding to the remaining rows of the same table). To do so, a new column was appended to each data set, named ANOMALOUS_BEHAVIOR with its classes description presented in table 4.5.

Table 4.5: Description of the classes used for labeling the collected data.

Label	Description
0	Non-anomalous samples. The vehicle is either braking with all the brakes correctly functioning or not braking at all.
1	Anomalous sample. The vehicle is braking and there is at least one brake failing.

After the labeling of each simulation data set all the samples were gathered thus forming the collected labeled data set. Table 4.6 presents the information of this data set.

Table 4.6: Collected data set information.

Label		Total Number of Samples	Size
0	1		
282716	16521	299327	262MB

4.3.3 Handling Class Imbalance

Whenever a data set contains a high disproportionate percentage of samples between different classes, it is assumed that the data set is imbalanced. Machine learning models are sensible and vulnerable to such issue, potentially causing poor classification performance whenever samples from the minority class are to be predicted. Such problem is caused by the model's biased learning process which favors the majority class due to its higher number of samples, thus disregarding the ones belonging to the minority class because of its reduced number of samples. We present different solutions proposed by the research community that we considered relevant for the mitigation of such issue.

- **Undersampling**: refers to the process of removing samples from the majority class in order to balance the classes distribution. This process however may eliminate important characteristics of the majority class by removing key samples and thus lead the model to potentially

misclassify future unseen samples of this class.

- **Random Oversampling**: works as opposite of undersampling, consisting on the duplication of random samples chosen from the minority class thus increase its total number of samples.

- **Synthetic Minority Over-sampling TEchnique (SMOTE)** [14]: this technique focuses on oversampling the minority class by creating new synthetic samples instead of merely duplicating existing ones. Briefly, new minority class samples are derived by taking each of this class samples and introducing synthetic samples along the line segments joining any/all of the k minority class nearest neighbors.

- **Adaptive Synthetic Sampling (ADASYN)** [33]: this techniques uses a weighted distribution for different minority class examples according to their level of difficulty in learning, causing more synthetic samples to be generated for minority class examples that are harder to learn compared to those which are easier.

- **Class Weighting**: this technique refers to the specification of different weights for each target class so that the model's loss function is given more or less penalty depending on its weight.

Because the different approaches presented require modification of the original data set, it was decided not to use a class imabalance handling technique forehand. Therefore for each approach presented, it is described the process for handling the class imbalance issue, explaining which techniques were applied, if any.

4.3.4 Normalization

The vast majority of machine learning models is highly sensitive to how the information fed as input is scaled. If different features have different scales, then contributions to the model's learning process will not be identical and thus causing it to be biased regarding up-scaled features. The normalization technique consists in a transformation of the input data, without disturbing or losing information, so that the different features of the data set are within a similar scale and thus contributing the same to the model's learning process.

The different machine learning models created for the brake anomaly detection task, presented in the subsequent sections, used normalization, more specifically, the *min-max* normalization for handling the data samples. This technique relies on a linear transformation of the data in order to fit all of the features' range into a common, defined boundary. For each feature x of a given sample, the following formula is applied for scaling:

$$x_{scaled} = \frac{(x - x_{min})}{(x_{max} - x_{min})} \tag{4.5}$$

4.3.5 Feature Selection

Feature selection is defined as the process of removing irrelevant, redundant and noisy features, in informal terms called columns, from its original set aiming to minimize the well-known dimensional complexity problem, casually known as the *curse of dimensionality* [5], which machine learning techniques remarkably struggle. Such problem is defined as the exponential complexity growth accompanying the number of dimensions (features) of the data, resulting in its sparsity and thus becoming problematic for methods, such as machine learning techniques, which require statistical significance.

The feature selection process carried in this work consisted on a brief overview on which features from the training data set were relevant to the problem by inspecting its type (only numerical values are allowed as input to machine learning models) and discarding any obvious non-relevant variables, such as the IDs and timestamps of the published ROS messages. Table B.1 presented in the Appendix B of this document, describes each of the features available in the data set obtained as part of the data collection process. Following a careful analysis of this features, the original data set feature space saw a dimensional reduction from forty-three to a twenty-one set of filtered features from which the model would depend on. The set of chosen variables is presented in table 4.7.

Table 4.7: Features selected for training of the machine learning models.

Feature	Type	Description
velocity	float	current velocity of the vehicle.
acceleration.linear.x	float	x component of the vehicle's linear acceleration.
acceleration.linear.y	float	y component of the vehicle's linear acceleration.
acceleration.linear.z	float	z component of the vehicle's linear acceleration.
orientation.x	float	x component of the vehicle's orientation.
orientation.y	float	x component of the vehicle's orientation.
orientation.z	float	x component of the vehicle's orientation.
orientation.w	float	x component of the vehicle's orientation.
control.brake	float	scalar belonging to {0.0, 1.0} representing the brakes activation.
control.throttle	float	scalar between [0.0, 1.0] representing the brakes activation.
control.steer	float	scalar between [-1.0, 1.0] representing the vehicle's steering direction.
pose.pose.orientation.x	float	x component of the current vehicle's orientation in quaternion form.
pose.pose.orientation.y	float	y component of the current vehicle's orientation in quaternion form.
pose.pose.orientation.z	float	z component of the current vehicle's orientation in quaternion form.
pose.pose.orientation.w	float	w component of the current vehicle's orientation in quaternion form.
twist.twist.linear.x	float	x component of the vehicle's linear velocity in free space.
twist.twist.linear.y	float	y component of the vehicle's linear velocity in free space.
twist.twist.linear.z	float	z component of the vehicle's linear velocity in free space.
twist.twist.angular.x	float	x component of the vehicle's angular velocity in free space.
twist.twist.angular.y	float	y component of the vehicle's angular velocity in free space.
twist.twist.angular.z	float	z component of the vehicle's angular velocity in free space.

4.4 One-Class Classification

Classification is recognized to be the task of identifying a class from which a sample belongs to. The most common and intuitive way of thinking about a classification problem is to have two classes from which a machine learning model must be able to efficiently differentiate and correctly classify previously unseen samples into one of the two target classes. Such process is known as binary classification and is later discussed in section 4.5. However, when collecting data from real-world samples, there may be cases where this data contains severe limitations, such as the significant absence of samples from one class (i.e. samples generated by rare events) which are difficult to synthetically generate and thus causing a serious class imbalance issue, or an infinite universe of samples impossible to be collected (i.e. normal emails in a *spam* detection model), which may result in a poor and flawed performance of the machine learning model due to its defected learning process, causing it to unable to differentiate samples from the distinct classes.

Schölkopf et al. [63] proposed a remarkable algorithm, adapting Support Vector Machines (SVM) to one-class classification, known as One-Class Support Vector Machines (OCSVM), or casually known as *novelty detection*. Such algorithm focuses on learning one class only by delimiting a frontier surrounding the distribution of the initial observations present in the training data and thus be able to conclude if a new sample is included within the delimiter frontier, assumed to be of the learned class, or not. This method was used for the current problem to understand how such approach is able to distinguish normal from anomalous samples and if it is suitable for use as the brake-specific anomaly detection model of an the autonomous vehicle.

4.4.1 Data Filtering

For the OCSVM algorithm previously described, it is required that its training data contains only samples from the one class to be learned. As described in section 4.3.2, the original data set collected from the various simulations recorded contained both normal and anomalous samples which were later submitted to a labeling process. Therefore, for this approach, there was the need of filtering the samples belonging to the class to be learned before feeding the data set to the algorithm. Such class could be one of the two labeled (normal or anomalous), however it was opted to select as target class the normal class as this presented a higher number of samples when compared against the minority (anomalous class) as it can be seen on the chart presented at the left of figure 4.7. By selecting the normal class samples for training of the algorithm, the boundaries delimited by it will be much broader and thus encompass a wider universe of samples presumably outperforming the results to be obtained if using the anomalous class to be learned.

The process of filtering the normal samples from the original training data set, involved the label column, known as `ANOMALOUS_BEHAVIOR`, to verify which of the samples satisfied the no-brake failure condition (label = 0). After such process was finished, an additional data set was created containing only normal samples, represented in the right chart of figure 4.7, and ready to be fed as input to the OCSVM algorithm. Note that because there is only one class, no class imbalance issue is present.

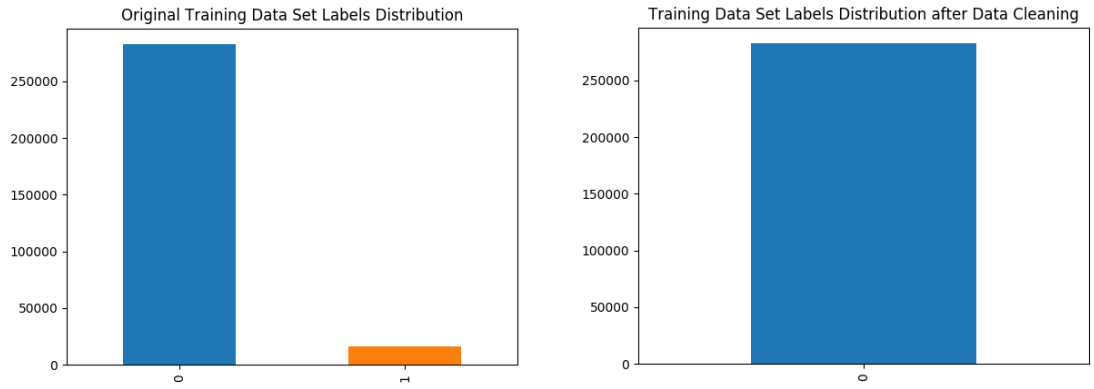


Figure 4.7: Label distribution transformation for the one-class classification model training data set.

4.4.2 Training and Primary Testing Phase

After the correct transformation of the data to be fed as input to the algorithm, this was ready to be split for both training and primary testing phases. As such, table 4.8 presents the portion assigned to each one of the phases.

Table 4.8: Original data set distribution for the construction of binary classification model.

Training	Test
80%	20%

The one-class SVM (OCSVM) algorithm supports a different number of mathematical functions, known as kernels (i.e. RBF, Sigmoid, Linear, Polynomial), which result in distinct learning rates and consequently training results. However, current literature has empirically proven that generally the RBF (Radius Basis Function) kernel outperforms the remaining ones. Bounsiar et al. [9] studied and compared the performance of multiple different kernels used for one-class classification concluding that only RBF kernels presented satisfactory results. Therefore, taking such results into consideration led to the decision of using this single kernel for the current classification problem. To better understand how such kernel is influenced by its *nu* hyperparameter, the algorithm was trained with five different values. Table 4.9 presents the primary testing values for each *nu* value, defined as the upper bound fraction of margin errors of the training data.

Table 4.9: One-class classification model hyperparameters tuning impact on accuracy.

Kernel	Hyperparameters				
	nu = 0.5	***nu* = 0.4**	*nu* = 0.3	*nu* = 0.2	*nu* = 0.1
RBF	39,20%	**46,49%**	60,19%	70,94%	82,55%

From the results presented, it can be concluded that the decrease of the *nu* parameter causes the model to *overfit* to the training data, narrowing the delimiting frontier over the trained samples therefore presenting better testing results. As such, to avoid selecting an *overfitting* model, it was opted to choose the algorithm which uses the RBF kernel and a *nu* value of 0.4. Following its selection, the model was ready to be submitted to the additional testing phase, production testing.

4.4.3 Production Testing

After selection of the model, production testing was conducted in order to test the model's efficiency in a live environment against previously unseen samples. Tables 4.10 and 4.11 present the calculated results for the production testing simulations previously described in section 4.1.3.

Table 4.10: One-class classification model RBF nu=0.4 production testing results.

	Normal				Anomalous			
	PPV	TPR	F_1	ACC	PPV	TPR	F_1	ACC
#1	0%	0%	0%	0%	-	-	-	-
#2	0%	0%	0%	25,30%	25,30%	100%	40,39%	25,30%
#3	0%	0%	0%	29,45%	29,45%	100%	45,50%	29,45%
#4	0%	0%	0%	48,22%	48,22%	100%	65,06%	48,22%
#5	0%	0%	0%	52,05%	52,05%	100%	68,47%	52,05%

Table 4.11: One-class classification model RBF nu=0.4 production testing units results.

	Normal				Anomalous			
	TP	TN	FP	FN	TP	TN	FP	FN
#1	0	0	0	8846	0	0	8846	0
#2	0	2112	0	6235	2112	0	6235	0
#3	0	2390	0	5725	2390	0	5725	0
#4	0	2345	0	2518	2345	0	2518	0
#5	0	6383	0	5880	6383	0	5880	0

A quick analysis on results obtained reveals that no prediction was accurate for the normal braking situation tested (#1). In fact, every prediction for each sample of the different simulations was predicted as a anomalous sample, as it can be observed by the nil values of both the true and false positives columns (TP; FP) of the "Normal" situation, presented in table 4.11. The increase of both models' accuracy and remaining metrics is result of its fully biased behavior in detecting each sample as anomalous, which are present in the anomalous simulations. In short, the accuracy of the model represents the percentage of anomalous samples (anomalous brakes) generated in the simulation.

This behavior was suspected to unwind. Remember that the OCSVM model works by defining a frontier around the samples learned during the training phase, in this case, samples from both normal and non-braking situations. Because there is an infinite number of such samples depending on the selected features (previously presented in table 4.7 of section 4.3.5), it was plausible to

expect that unseen samples were to be classified as anomalous, as these although non-anomalous, would probably have very different feature values (i.e. velocity, orientation.x) from the original training data samples. This would result in them falling outside of the delimited frontier's internal region thus leading the model to incorrectly classify them as anomalous.

The obtained results clearly reveal the lack of learning skill by the algorithm with this training data as it is completely biased in predicting every unseen sample as anomalous. This model is then concluded to be very inefficient for the context of the problem and is automatically rejected to become the brake-specific anomaly detection model, as its use may result in disastrous outcomes such as preventing the vehicle to start due to the current anomalous detection component control action (activation of the hand brake every time an anomaly is detected).

4.5 Binary Classification

Binary classification is a frequent type of classification when applying machine learning classification techniques due to its intuitive reasoning: classification of a given sample into one of two distinct classes. The machine learning model goal is to learn the relevant characteristics of the training data so it is able to correctly classify future, unseen samples.

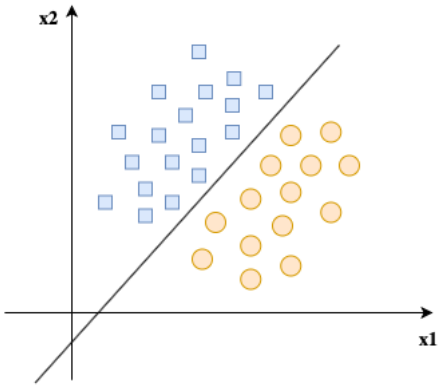


Figure 4.8: Binary classification.

4.5.1 Model Architecture

We decided to use deep learning to implement a robust model capable of distinguishing normal from anomalous situations. In abstract, this concept can be described as a particular kind of machine learning which enables computers to efficiently learn complex concepts through the combination of simpler ones [28]. A neural network is the underlying concept of deep learning consisting on a machine learning model composed by a dense network of interconnected processing nodes, known as neurons, responsible for mapping input values to a single output value through the use of an activation function. Commonly, such neurons are organized into feed-forwarding units of layers, meaning that the combined output of the neurons of each layer will be used as input values

for each of the next layer's neurons. The neural network layers are defined as input layer, responsible for receiving the data features, hidden layers, responsible for processing the information, and the final output layer, responsible for returning the processing result.

The neural network created for this task is composed by an input layer of twenty-one neurons, representing the features to be fed as input to the model, four hidden layers for the processing of the features and the output layer, making use of the softmax activation function, responsible for returning a vector of probabilities for each target class (normal or anomalous). Overall, this model consists on six different layers to evaluate the incoming generated samples as either normal or anomalous. Figure 4.9 present a visual representation of the created neural network for this classification task.

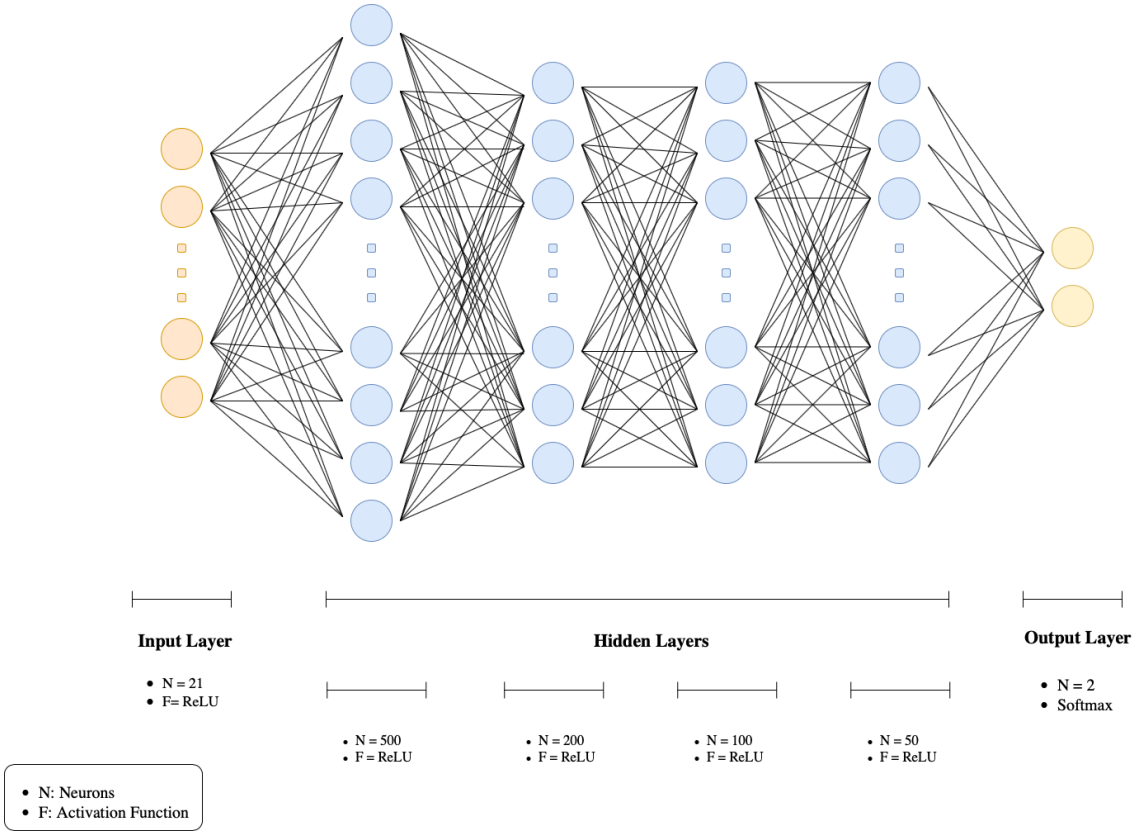


Figure 4.9: Architecture of the deep neural network created for binary classification.

4.5.2 Data Labeling

Recalling the presented explanation of the data labeling process in section 4.3.2, supervised learning techniques, such as the current, require the presence of labels for guiding the model into learning each class's characteristics for future correct classification. As such, in order for the binary classification deep learning model to be able to learn the distinct classes' characteristics, labels should be present for each of the samples presented in the training data set. This process

had already been executed upon parsing the recorded collected data files therefore there was no need of re-labeling the original data set as this already included the correct labels for both possible situations: normal braking/non-braking situations and anomalous braking situations.

4.5.3 Class Imbalance

A quick analysis of the label classes distribution revealed an explicit class imbalance issue. Figure 4.10 presents two charts representing the distribution of the target/label classes. At the left, the original data set label class distribution is presented, revealing a clear evidence of imbalance between the two distinct classes. The number of normal samples present in the data set is significantly higher than the one for the anomalous samples. As explained in section 4.3.3, this may cause severe issues in the model's learning process as it will be biased in learning the characteristics of the majority class data, here represented as samples from normal situations, and disregard the characteristics of the minority class, also known as anomalous braking situations. Therefore, in order to solve this issue and balance the classes' distributions, the SMOTE technique was used in pair with class weighting.

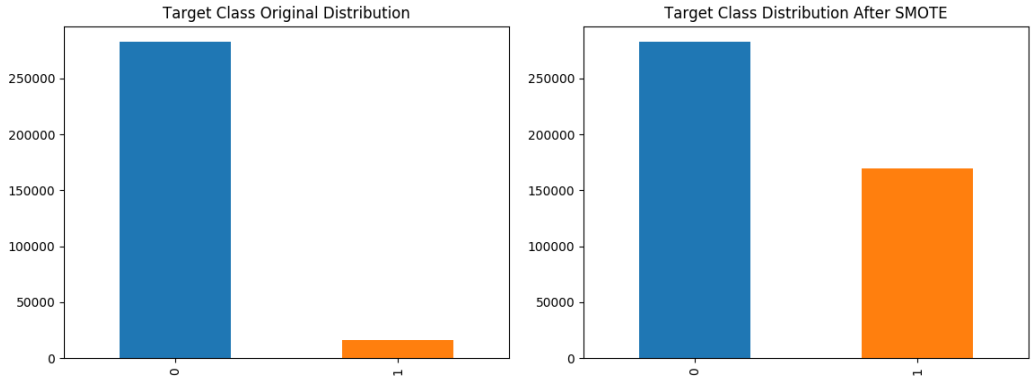


Figure 4.10: Original class distribution for the binary classification model training data set.

New synthetic samples were generated using the SMOTE technique, increasing the number of samples from the minority class up to 60% of the number of samples of the majority class thus balancing the distribution between both of them. Note that this percentage was chosen taking into account the synthetic sample generation process described in 4.3.3, which derives new samples from the closest k-neighbors which may overlap with normal braking samples due to the close values that these share. This may lead the model into confound the two distinct braking situations. The chosen 60% percentage was intended to represent a value large enough to impact the anomalous class learning process yet to not shadow the normal braking samples. The rightmost chart presented in figure 4.10 exposes the distribution of the label classes after this technique was applied.

Apart from the generation of synthetic samples, different class weights were specified for each target class, as presented in table 4.12.

Table 4.12: Class weights specified for the target classes of the binary classification model.

Target Class	Class Weight
0	1.0
1	2.0

4.5.4 Training and Primary Testing Phase

After the successful generation of the synthetic examples from the minority class, the data set was prepared to be given as input to the neural network. To better understand the relations between the features of the data set, figure B.3 of Appendix B presents the correlation matrix of these. Before feeding the data for training of the model, it was necessary to split the data into different portions, as presented in table 4.13.

Table 4.13: Original data set distribution for the construction of binary classification model.

Training	Validation	Test
60%	20%	20%

The model was then ready to start training and learning how to differentiate between both of the target classes. Table 4.14 presents the primary testing results with regard to the different combinations of hyperparameters selected.

Table 4.14: Binary classification model hyperparameters tuning.

	Hyperparameters			Class Weights		Metrics*		
	Epochs	Batch	*lr*	0	1	PPV	TPR	F_1
A	50	50	0.002	1.0	2.0	84,16%	86,09%	82,73%
B	50	50	0.01	1.0	2.0	18,79%	50,00%	27,31%
C	100	200	0.001	1.0	2.0	86,56%	88,98%	86,45%
D	**100**	**200**	**0.002**	**1.0**	**2.0**	**87,46%**	**89,87%**	**87,75%**
E	100	200	0.005	1.0	2.0	83,95%	86,06%	82,47%

*Macro Average

Multiple hyperparameters combinations were tested in order to optimize the results obtained. From the information presented in table 4.14, it can be concluded that the model is sensitive to the chosen learning rate, achieving better results when this is low. A higher learning rate causes the model to diverge and thus lessen its performance and robustness. One more concluding note is that smaller epochs and batch sizes also decrease the model's performance when compared to higher values with the same learning rate, as it is proven from model A and D. Therefore, a general conclusion is that best results are achieved for this model when the learning rate is small but both the number of epochs and batch size is large.

From the set of results obtained, the optimal learning rate for the tested hyperparameters combinations is 0.0001 together with 100 epoch and a batch size of 200, presented in row D of the same table. Therefore, this hyperparameters combination was selected for the production testing phase. Figure B.1 of appendix B presents two charts representing the accuracy and loss for both training and validation results during the learning process of this binary classification machine learning model.

4.5.5 Production Testing

Upon selecting the best hyperparameters combination for the model, the production testing phase was initiated in order to understand if the model is subject to any particular issue, such as the biased behavior of the one-class classification model previously presented. Table 4.15 and 4.16 presents results obtained for each simulation run.

Table 4.15: Binary classification production testing results.

	Normal				Anomalous			
	PPV	TPR	F_1	ACC	PPV	TPR	F_1	ACC
#1	100%	80,88%	89,43%	80,88%	-	-	-	-
#2	97,80%	99,98%	98,88%	98,41%	99,95%	94,71%	97,26%	98,41%
#3	84,48%	99,83%	91,52%	87,68%	99,47%	63,48%	77,51%	87,68%
#4	99,99%	99,92%	99,96%	99,94%	99,82%	99,99%	99,90%	99,94%
#5	93,90%	99,77%	96,75%	95,02%	99,21%	81,35%	89,40%	95,02%

Table 4.16: Binary classification production testing units results.

	Normal				Anomalous			
	TP	TN	FP	FN	TP	TN	FP	FN
#1	21771	0	0	5146	0	21771	5146	0
#2	23161	9311	520	5	9311	23161	5	520
#3	15466	4940	2841	26	4940	15466	26	2841
#4	17132	7337	1	13	7337	17132	13	1
#5	12374	3508	804	28	3508	12374	28	804

From the results presented for the non-anomalous braking system (#1), we concluded that the model is somewhat biased in predicting some samples as anomalous, more precisely 5146 samples out of 26917. In the remaining simulations (#2-#5), defined as anomalous simulations, the model was able to correctly classify the vast majority of the samples for both classes, showing worst results in simulation #3 and best in simulation #2. Such results are a warning sign that the model may be biased towards classifying each braking sample as anomalous, independently of its situation. This explains the number of false positives predicted in the normal simulation and the lower values of false negatives predicted in the anomalous situations.

Overall, it can be concluded that this model achieved satisfactory results, not forgetting that its generalization to previously unseen data was sublime, demonstrating better results in the production testing phase than the ones obtained in primary testing. Even though it presented such results, it is inferred that the model is moderately biased into predicting any braking sample as anomalous. Nevertheless, it outperformed the one-class classification model making it a possible candidate for integrating the anomaly detection component.

4.6 Multi-Class Classification

This type of classification task attempts to classify a sample in a universe of more than two possible classes (binary classification). Since the problem is to detect brake-specific anomalies, this approach intents to differentiate between three distinct classes contrasting to the two previously presented approaches: one-class and binary classification. With multi-class classification, our goal is to distinguish three possible scenarios: non-braking, normal braking and anomalous braking samples. By doing so, there is more confidence that the model will specifically improve the learning process to differentiate between normal and anomalous braking samples comparing to the previous approaches, and thus provide more accurate predictions.

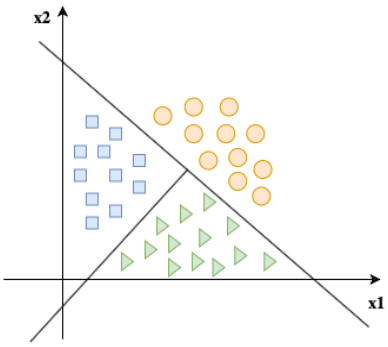


Figure 4.11: Multi-class classification.

4.6.1 Data Re-Labeling

Training data to be fed as input to the multi-class classification model requires a slight transformation of the labels present in the original data set. Instead of having the original data set labeled into two different classes, normal and anomalous, this approach requires sample labels to be categorized in one of three different classes, described in table 4.17.

The proposal of three classes to be learned how to be differentiated, leads the neural network to more clearly distinguish normal *versus* anomalous braking scenarios as opposite to the previously presented binary classification approach. After the re-labeling process was completed, the data set then contained three separate classes to be learned by the model. Figure 4.12 presents their distribution.

Table 4.17: Description of the classes used for labeling the collected data.

Label	Description
0	Non-braking samples. The vehicle is not braking.
1	Well-functioning braking samples. The vehicle is braking and there is no anomaly on the brake system.
2	Malfunctioning braking samples. The vehicle is braking and there is at least one brake failing.

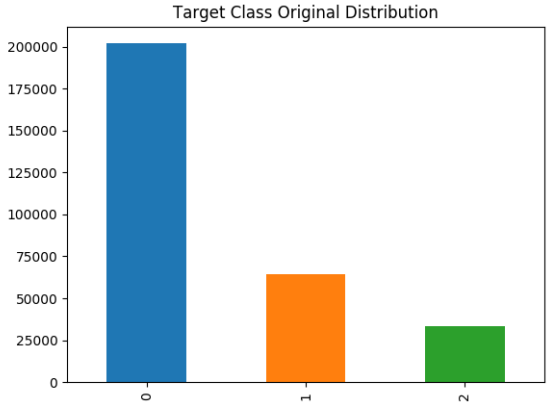


Figure 4.12: Class distribution for the multi-class classification model training data set.

4.6.2 Class Imbalance

Although there is obvious presence of imbalance between the target classes, it was decided not to apply any oversampling method to the input data due to the risk of overlapping both normal and anomalous braking samples which have now its own separate target classes as opposite to the binary classification approach. Therefore, only class weighting was used to handle the class imbalance issue in order to not mix information between the different target classes. The weights specified for each class are presented in table 4.18.

Table 4.18: Class weights specified for the target classes of the multi-class classification model.

Target Class	Class Weight
0	1.0
1	1.5
2	2.0

4.6.3 Model Architecture

We decided to use a deep learning model similar to the one used for the binary classification task due to its satisfying results. By doing so, there is also a better understanding on how similar neural

networks behave when learning the collected information. This neural network is composed by an input layer of twenty-one neurons, four hidden layers for information processing, composed by the same neurons as the one for binary classification, and a three-neuron output layer representing the probabilities values of each target class. Figure 4.13 represents a visual diagram of this deep neural network created.

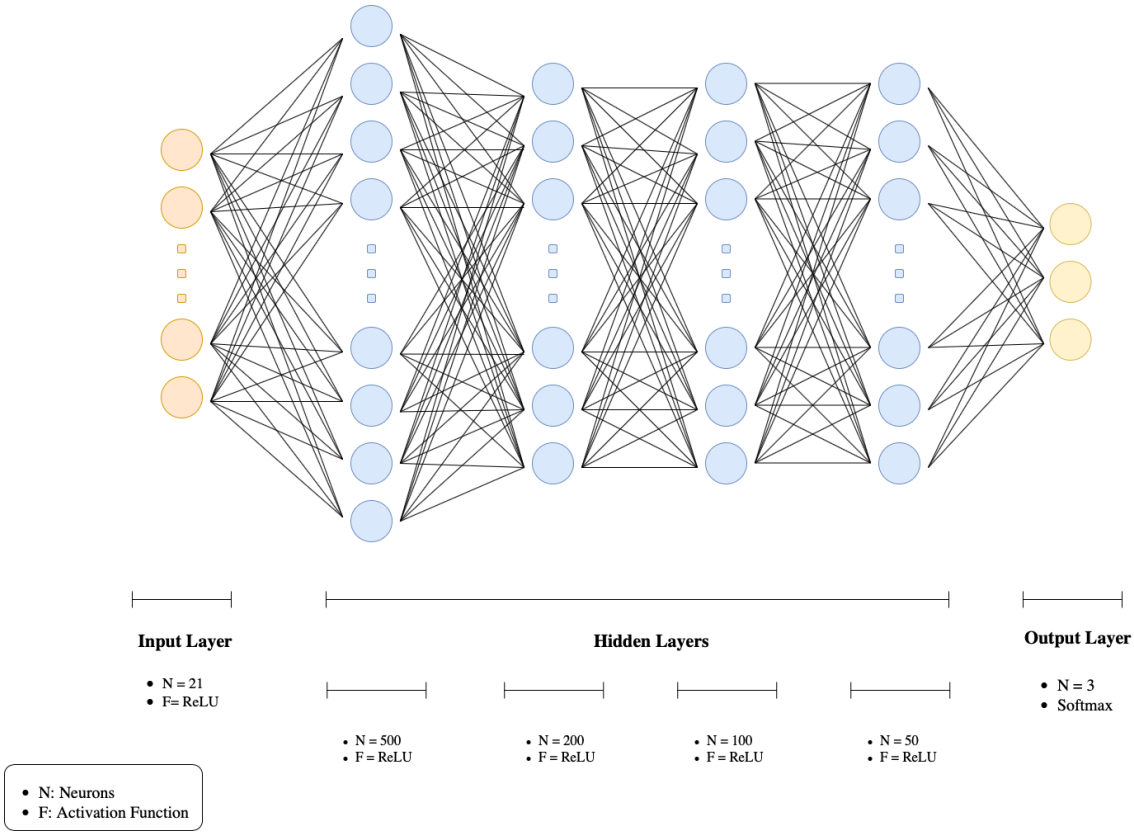


Figure 4.13: Architecture of the deep neural network created for multi-class classification.

4.6.4 Training and Primary Testing Phase

Figure B.4 of Appendix B presents the correlation matrix of the features of the input data set. Table 4.19 presents the distribution of the data set used in this approach.

Table 4.19: Original data set distribution for the construction of binary classification model.

Training	Validation	Test
60%	20%	20%

From the training metrics presented, it is concluded that a smaller learning rate, in addition to a larger value of epoch and batch size, leads to better results as it is presented in model A,C,F,G. The model tested with both the lowest epoch and batch size (model E) presented significant lower results when compared to the others. The model with the lowest learning rate (model G) together

Table 4.20: Multi-class classification model hyperparameters tuning.

	Hyperparameters			Class Weights			Metrics*		
	Epochs	Batch	*lr*	0	1	2	PPV	TPR	F_1
A	100	200	0.001	1.0	1.5	2.0	90,02%	84,16%	85,63%
B	100	200	0.003	1.0	1.5	2.0	85,84%	87,25%	86,30%
C	150	200	0.001	1.0	1.5	2.0	88,90%	87,91%	88,34%
D	150	200	0.002	1.0	1.5	2.0	89,14%	85,02%	86,22%
E	50	50	0.001	1.0	1.5	2.0	82,10%	80,53%	80,34%
F	150	300	0.002	1.0	1.5	2.0	90,55%	87,67%	88,70%
G	**150**	**300**	**0.0001**	**1.0**	**1.5**	**2.0**	**90,52%**	**89,39%**	**89,88%**

*Macro Average

with the highest number of both epochs and batch size presented the most satisfactory metrics outperforming the recall and F_1-Score of every other hyperparameter combination, falling behind a mere 0.3% on model's PPV value. Therefore, the selected model from the different tested hyperparameter combinations was G as it obtained the best results overall and thus yields more confidence compared to the others, when used in a live environment. Figure B.2 of appendix B presents both the results the model's accuracy and loss when in the training and validation phases per epoch.

4.6.5 Production Testing

The multi-class model was submitted to the same conditions as the ones presented for the previously described approaches. Table 4.21 and 4.22 presents the results obtained for the production testing simulations of this model.

Table 4.21: Multi-class classification production testing results.

	Not Braking				Normal Brake				Anomalous Brake			
	PPV	TPR	F_1	ACC	PPV	TPR	F_1	ACC	PPV	TPR	F_1	ACC
#1	100%	100%	100%	100%	100%	96,39%	98,16%	98,99%	-	-	-	-
#2	100%	99,98%	99,99%	99,99%	-	-	-	-	99,77%	18,08%	30,61%	71,01%
#3	100%	99,97%	99,98%	99,98%	-	-	-	-	99,86%	38,92%	56,01%	78,01%
#4	100%	99,99%	100%	100%	-	-	-	-	99,98%	36,72%	53,72%	70,69%
#5	100%	99,78%	99,89%	99,85%	-	-	-	-	98,86%	45,21%	62,04%	83,71%

Table 4.22: Multi-class classification production testing units results.

	Not Braking				Normal Brake				Anomalous Brake			
	TP	TN	FP	FN	TP	TN	FP	FN	TP	TN	FP	FN
#1	23378	9076	0	1	8748	23379	0	328	0	32126	329	0
#2	17815	9752	0	4	0	19582	7989	0	1763	17815	4	7989
#3	19746	11095	0	6	0	24070	6777	0	4318	19746	6	6777
#4	15491	13367	0	1	0	20401	8458	0	4909	15491	1	8458
#5	23413	9793	0	51	0	27891	5366	0	4427	23413	51	5366

For the normal braking simulation, represented by #1, it can be seen that the model obtained very decent results in distinguishing both normal brakes from non-braking situations, with an accuracy of 100% for non-braking samples classification and an extraordinary accuracy of 98,99% for normal brakes samples classification. It can be observed that mere 328 samples were predicted as anomalous (false positives), thus obtaining the best results for this production testing run when compared to the same simulation results of the previous approaches.

When analysing the results of the different anomalous situations, it is clear that the metrics calculated for the anomalous brakes are far below the expected. It can be seen that they reveal surprisingly low recall (TPR) results causing the F_1-Score to drop. When analysing the units results, it is concluded that there is a high number of false negatives (FN) for the anomalous brake which are classified as normal braking samples. Note that, the recall percentage and consequently both F_1 and accuracy, increases with the the level of severity of the brakes failures. This behavior was expected as the severity of the brakes failing will influence the effect produced on the vehicle's braking operation and thus the sensor values for prediction will be closer or distant from the ones of the original braking situation depending on the anomalous severity. The results for such situations were not deemed to be satisfactory. A deeper analysis of the results obtained for the production testing simulation runs demonstrates that the model is able to correctly predict normal braking samples but failing when classifying braking samples originated from anomalous scenarios. To find the root cause of such problem, an additional simulation run was conducted, submitting the model against to one more anomalous scenario, so that the predictions of the model were analysed depending on the vehicle's current operation. Figure 4.14 presents a chart of the samples generated for this additional run conducted by the Carla simulator's autopilot mode, submitted to the same conditions as production testing #5 runs (all brakes failing). The chart relates three different variables, the vehicle's (`velocity`), the model's predictions (`PREDICTION`) and the labels column (`ANOMALOUS_BEHAVIOR`). Note that because this was an anomalous run, the post-simulation labeling process was to define all the braking samples as anomalous (label = 2) and the remaining as normal (label = 0).

From the presented values, it is concluded that this simulation was subject to a total of three stops (`velocity` = 0). When taking a closer look at each, there is a clear discrepancy between each sample's label (`ANOMALOUS_BEHAVIOR`) and its predicted value (`PREDICTION`). Taking the second brake in consideration ($x \in [3200, 4200]$), it can be seen that the model correctly predicts the braking samples as anomalous (`PREDICTION` = 2) while its `velocity` does not reach zero, presented by the green spike near $x = 3200$. However, when the car reaches its full stop the model starts predicting such samples as normal brakes instead (`PREDICTION` = 1), represented by the different values of both the orange and green lines. When analysing the third braking situation, its values are identical. Therefore, it is reasonable to assume that the model is biased in predicting anomalous braking samples as normal, whenever the vehicle is not moving (velocity = 0). This means that the model is actually predicting correctly the majority of the anomalous samples during the vehicle's movement but failing to do so when this is stopped. Such problem questions the model's current operation in understanding if it is required to detect anomalous

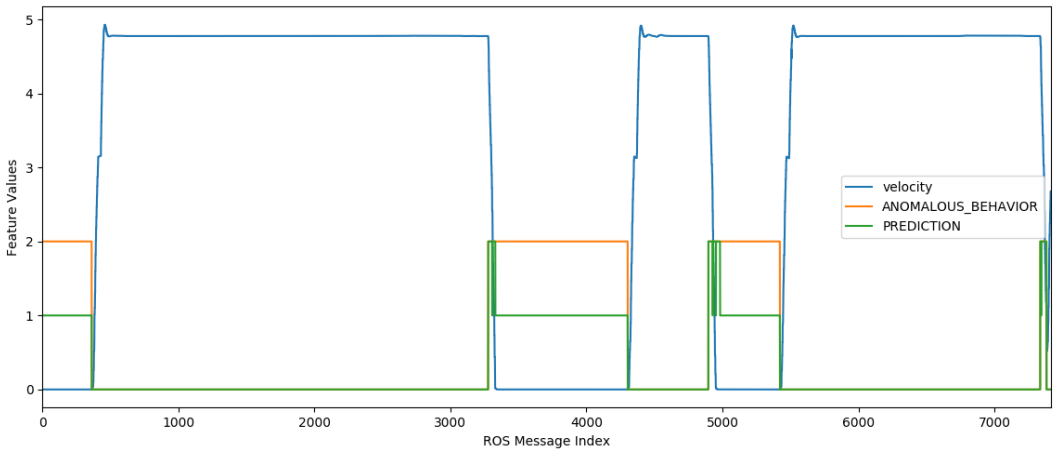


Figure 4.14: Relation between the model prediction and the velocity of the vehicle in an anomalous situation.

braking samples when the car is not moving (velocity = 0). This is left for a more profound exploration as part of future work as there is confidence that the model will be able to leverage its performance by discarding any braking samples of a non-moving vehicle.

4.7 Comparative Analysis

Following the creation of the different brake-specific anomaly detection models previously described, it was required to analyse and compare their performance against throughout the different testing phases they were submitted to.

The first model presented, one-class classification, demonstrated low testing results with a mere 46,49% of accuracy in the primary testing phase thus revealing a non-satisfactory performance for an anomaly detection mechanism. Furthermore we concluded that its performance fell behind when compared with the posterior approaches hence its rejection to integrate the anomaly detection component. Because the models created for the remaining approaches rely on deep learning techniques, each primary testing results are presented in 4.23 with regard to the hyperparameters combination defined.

Table 4.23: Binary *versus* multi-class primary testing results comparison.

	Hyperparameters			Class Weights			Metrics*		
	Epochs	Batch	*lr*	0	1	2	PPV	TPR	F_1
Binary	100	200	0.002	1.0	2.0	-	87,46%	89,87%	87,75%
Multi-Class	150	300	0.0001	1.0	1.5	2.0	90,52%	89,39%	89,88%

*Macro Average

We concluded that each model obtained similar results although different hyperparameters combinations were used. The multi-class classification model presented better results overall, however it presented a smaller learning rate and a bigger epoch and batch size, possibly resulting in a better understanding of the data.

In order to compare the results obtained by the additional conducted testing phase, known as production testing, the different simulations results were aggregated into different charts, one per each simulation run. The results presented for each model were derived by applying formula 4.6 which calculates the arithmetic mean of the results obtained for each target class per available metric (PPV, TPR, F_1, ACC). Note that for both one-class and binary classification simulations in which there is no anomalous samples (simulation #1), we use $N = 1$ because there exists no true positive values and thus calculating those metrics would not be relevant for this analysis.

$$Metric_{mean} = \sum_{N} \frac{Metric_{Class}}{N}, \quad N = Number\,of\,Classes \tag{4.6}$$

Figure 4.15 presents five distinct charts, one per each production testing simulation run, so that a visual representation of the results is provided and thus better understand each of the models' performance when submitted against the different production testing scenarios.

From the presented charts, there is clear evidence that both the binary and multi-class classification models outperform the one-class classification by a large margin. As previously described, such model was not able to correctly classify previously unseen samples as normal, demonstrating poor results throughout the different phases. Again, this model is immediately discarded for integrating the anomaly detection component.

A deeper analysis focusing on the binary and multi-class classification models reveals that both demonstrated similar performance results in simulations #1 and #3, however, the multi-class model failed to outperform the binary one in the remaining simulations. As previously discussed, the multi-class model is biased into classifying every braking sample of a fully stopped car as a normal brake, which is causing the model's performance to fall behind when compared against the results of the binary classification model. However, one must take into account that the latter is believed to be suffering from a more serious issue consisting on an existence favoring of predicting any braking samples as anomalous thus proving the better results obtained in the anomalous production testing scenarios but its worse performance, when compared to the multi-class model, in the normal braking scenario (#1).

We are confident that the multi-class model performance metrics obtained in anomalous situations can be drastically improved by only feeding to it samples generated whenever the vehicle is moving. By doing so, it is expected to decrease the level of false positives of the normal braking class during anomalous simulation, which, as demonstrated in section 4.6, corresponded to the model's predictions whenever the car was fully stopped. Note that such behavior is not assumed to affect the security provided by this anomaly detection mechanism as in a real-world scenario it would be difficult to detect a vehicle's brake anomaly when stopped. However, such approach is left for future development as there has to be an extensive study on its implications as well as the

model's adaptation to it.

4.8 Model Selection

We found that multi-class classification model outperformed the remaining approaches by providing a higher level of confidence of its correct behavior when distinguishing normal from anomalous braking samples (of a non-stopped vehicle) as well as outstanding results when detecting samples generated from non-braking situations. This model was selected to integrate the anomaly detection component and test its robustness against adversarial learning attacks.

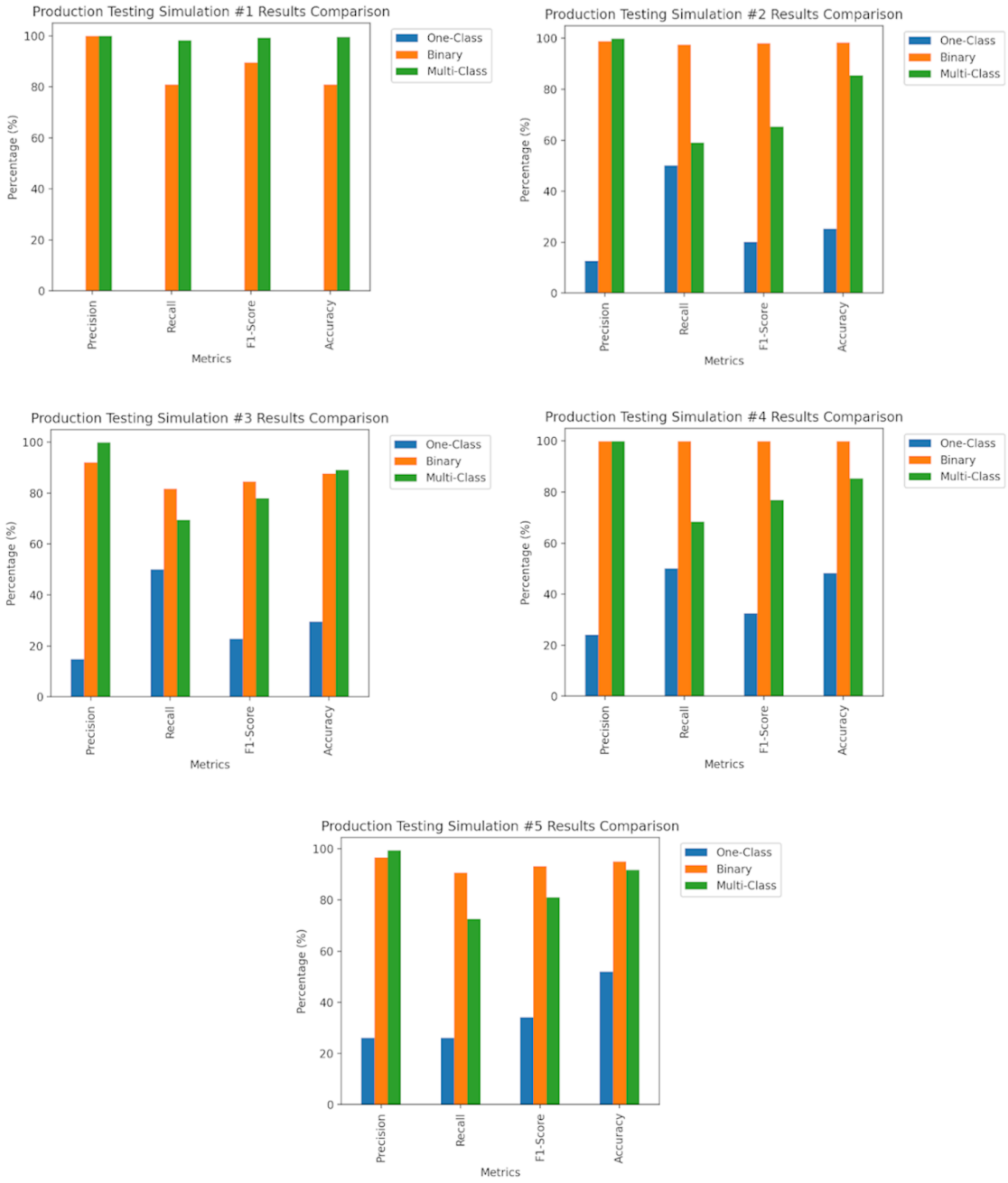


Figure 4.15: Charts of the metrics for comparison of the implemented approaches: one-class, binary and multi-class classification.

Chapter 5

Attacking the Anomaly Detection Component

This chapter presents the attacking strategies followed to simulate an adversarial learning attack to an autonomous vehicle, more specifically, an attack which goal is to exploit the brake-specific anomaly detector, presented in chapter 4, so that anomalous samples become misclassified as normal and thus could result in an accident.

The first section of this chapter presents different adversarial learning attacks conducted against the anomaly detection model and how these impacted its accuracy. Afterwards, an in-depth description of a possible attack to the ROS framework is presented, exploiting a vulnerability permitting malfeasants to manipulate and intercept communication between two or more ROS nodes. Such attack provides an example of ROS vulnerabilities and contextualizes the reader to understand how ROS systems, and consequently the created anomaly detection component presented in chapter 3, can be exploited using this attack together with adversarial learning techniques in order to cause system malfunctioning. The final section of this chapter focuses on describing the set of experiments conducted using the simulation system to compare the effects of a brake-specific anomaly detection component in both safe and exploited situations.

5.1 Adversarial Samples Generation

The generation of adversarial samples can be unwound through the application of various attacking techniques subject to different assumptions. For white-box adversarial learning techniques it is implied that the attacker has knowledge about the model's internals, as opposite to black-box techniques which assume that the attacker does not have any access to the model's internals nor its training data, making them harder and slower to execute. We used both white-box and black-box techniques to execute such attacks, relying on the *adversarial robustness toolbox* framework [51].

Due to the high number of existing adversarial learning attacks and to better understand how these affect the model's ability to correctly predict anomalous samples selected state-of-the-art attacks were executed against the developed multi-class brake-specific anomaly detection model.

We evaluated the model's robustness against three untargeted white-box attacks (Fast-Sign Gradient Method (FGSM) [30], Basic Iterative Method (BIM) [38], Carlini & Wagner (L_∞) [12]), presented each results in table 5.1, and an additional untargeted black-box attack (Zeroth-Order Optimization (ZOO) [17]), presented in table 5.2. Note that it is outside of the scope of this work to analyze the internals of each attack.

Table 5.1: White-box attacks impact on the accuracy of the model.

	ε*					
	-	0.1	0.2	0.3	0.5	0.7
FGSM	75,61%	28,16%	28,30%	28,50%	29,77%	29,68%
C&W	75,61%	29,52%	28,90%	28,89%	28,54%	28,94%
BIM	75,61%	29,33%	29,33%	29,33%	29,33%	29,33%

*ε: perturbation of the attack (higher values correspond to higher perturbations).

Table 5.2: Black-box attack impact on the accuracy of the model.

	lr
	0.01
ZOO	24,39%

**lr*: learning rate.

We concluded that the white-box attacks applied to the model had a similar impact on its accuracy with regard to the tested perturbation (ε) values. The model's accuracy dropped $\sim$46% proving their success and their threatening characteristics against the robustness of machine learning models. The state-of-the-art black-box attack ran against the model obtained better results when compared to the white-box attacks, causing a decrease of 51,22% of the model's accuracy.

The set of attacks conducted against the brake-specific anomaly detection model proves that these impose a high-level threat against the model's capability to correctly distinguish incoming samples between the different target classes and thus menace the correct operation of the autonomous vehicle security system, possibly resulting in devastating outcomes.

5.2 Architecture of the Attack

The development of an attack targeting an autonomous vehicle system is a complex and sedulous task which requires considerable knowledge about the system's operation. In this work, an autonomous vehicle system is simulated using both the Carla simulator and ROS, to which the latter is subject to numerous attacking techniques targeting different components of the ROS computational graph. This section presents the explanation of one of such attacks which focuses on manipulating the communication between two nodes in order to contextualize the reader to how it can be executed, in particular, against the novel ROS component proposed in chapter 3, the anomaly detection component.

5.2.1 Description

As described in chapter 2, ROS is a framework well-known for its ability to create robotic applications but also because of the multiple severe security vulnerabilities it contains, as opposite to the new enhanced version ROS2 [71], making it a careful-to-use tool for non-private applications. Among many of these vulnerabilities two critical features are absent, the lack of authentication and authorization mechanisms in the inter-process communication used in the framework's main operation process, in other words, for communication between ROS nodes. As such, Dieber et al. [22] explored some of the existing vulnerabilities and proposed a data injection attack, known as the "*Stealth Publisher Attack*", presenting its detailed description. This attack targets a subscriber node by isolating it from one or more publisher nodes, and then tricking it to initiate a new topic communication to an unauthorized publisher. The publisher is then able to send data to the vulnerable node impersonating a regular, authorized publisher node of this topic. To better contextualize the reader to such attack and to explain how this could be used for exploiting the created anomaly detection component presented in chapter 3, a detailed description of its process is presented. The attack first focuses on the retrieval of the target nodes information, known as the information retrieval phase, followed by the attack itself or attacking phase.

Figure 5.1 presents a visual representation of the attack to the anomaly detection component created in chapter 3 in order to not only present a sequence diagram of the attack workflow but also the component's inerent vulnerabilities.

The attacker first starts by querying the master node, invoking its `getSystemState` API (1), in order to retrieve information about the current system state of the ROS network and thus gain full knowledge on the current active topics and their publisher/subscriber nodes. After selecting the target publisher/subscriber nodes, it re-queries the master node, this time calling its `lookupNode` API twice (2,3), in order to retrieve the XML-RPC URIs of both of these nodes. With this information, the attacker can now pass to the second phase of the attack.

Having the target nodes information, the attacker is now able to directly communicate with each one. Therefore, it impersonates the master node by calling the `publisherUpdate` method of the subscriber node's slave API with its own identity (URI) as the only publisher node of the specified topic, causing an update of the list of the subscriber's known publishers for the given topic (4). Upon this, the subscriber will terminate communication with the publisher node and initiates one with the attacker instead by sending a `requestTopic` call, which it will reply with a list of its support protocols for communication (5). Note that ROS support at least two different protocols for data communication (TCPROS or UDPROS) and therefore the attack differentiates between the one used. For a TCPROS data transport, the attacker forwards the received call to the publisher, receiving as result a message containing the host and the port where the publisher listens for new TCP connections, among others. Upon receiving this, it first alters this message by changing host and port information to its own and then forwards it to the subscriber (6). Next, the subscriber establishes a TCP connection with the attacker by sending a TCPROS header message which is again forward to the publisher in order to retrieve a well-formatted reply to forward to

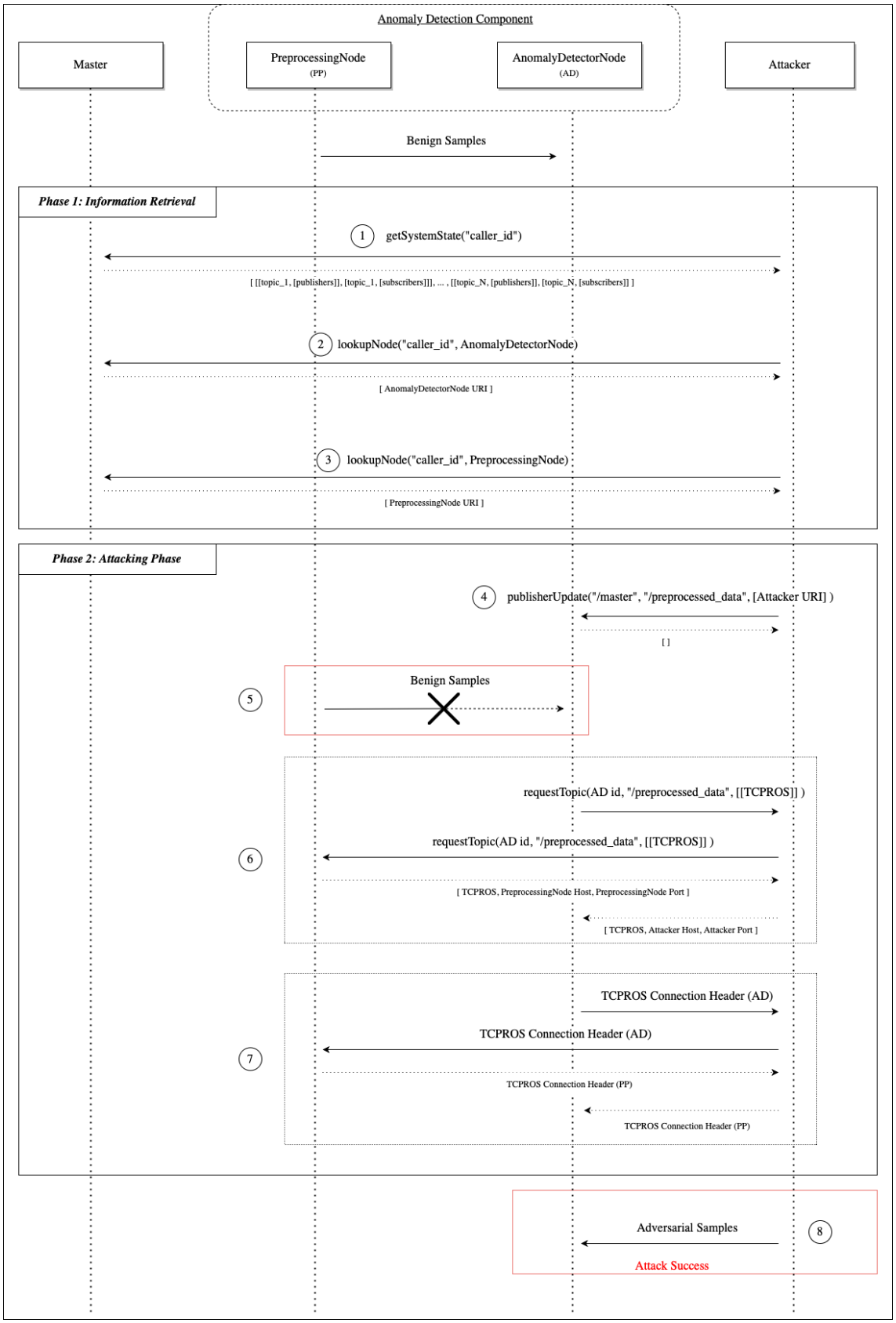


Figure 5.1: Adapted sequence diagram of the *Stealth Publisher Attack* to the proposed anomaly detection component.

the subscriber node (7). At this point, the attacker is free to send its own data messages to the subscriber node (8), which in this case corresponds to the adversarial learning generated samples.

5.2.2 Practical Approach

The implementation of the described attack was outside of the scope of this work, therefore a simpler approach was chosen to simulate such attack, consisting on using the adversarial samples generated by the FGSM attack. Each time the anomaly detection component received a braking sample it was discarded and then replaced by a random selected adversarial sample which was fed as input for its prediction. This approach avoid having to implement the ROS-specifc attack yet achieving the same outcome which was to evaluate the performance of the brake-specific anomaly detection model when presented against adversarial learning samples.

5.3 Experimental Setup & Observed Results

Following the successful generation of the adversarial learning samples, result of the previously presented techniques, it was time to attack the brake-specific anomaly detection model with these in order to understand how these impact the vehicle's operation. To do so, three simulations, described in table 5.3, were conducted, each representing the possible braking scenarios described throughout this work.

Table 5.3: Description of the different simulations ran for testing the studied situations.

Description	Brakes Failing	Driving Mode
No Anomaly Detection	FL, FR, RL, RR	Manual
Attack-Free Anomaly Detection Model	FL, FR, RL, RR	Manual
Attacked Anomaly Detection Model	FL, FR, RL, RR	Manual

The goal of running and comparing such simulations is to first understand how the created anomaly detection model can prevent the vehicle's enrollment in unwanted dangerous situations, thus revealing its importance in the enhancement of an autonomous vehicle system security, and in second to understand how such attacks can take advantage of the model's vulnerabilities to originate such dangerous situations. Note that these experiments were ran using the Carla simulator manual mode, as the available autonomous driving agent of Carla-ROS bridge, given an initial and ending point, generates a random route between them. This was not ideal as each of the simulation's vehicle route would most likely be different and thus their comparison would not be of much relevance. Therefore, these simulations were manually conducted in a straight road containing an obstacle at its end. Figure 5.2 presents a diagram representing the three experiments ran and their expected outcomes.

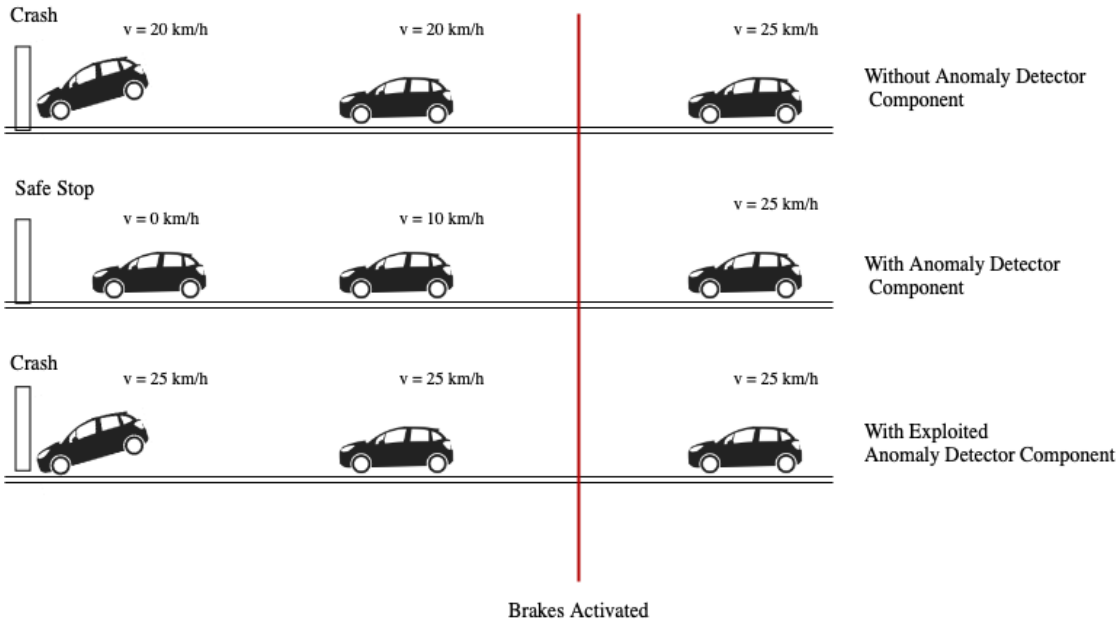


Figure 5.2: Representation of the different simulations ran for testing the impact of the anomaly detector component.

5.3.1 No Anomaly Detection Run

The first experiment consisted on recording a simulation ran without the activation of the anomaly detection component in order to understand the vehicle's original behavior when in an endangered situation. Figure 5.3 presents a chart with information corresponding to this described simulation run relating three different features, `velocity`, `control.brake` and `control.hand_brake`, relevant for analysis.

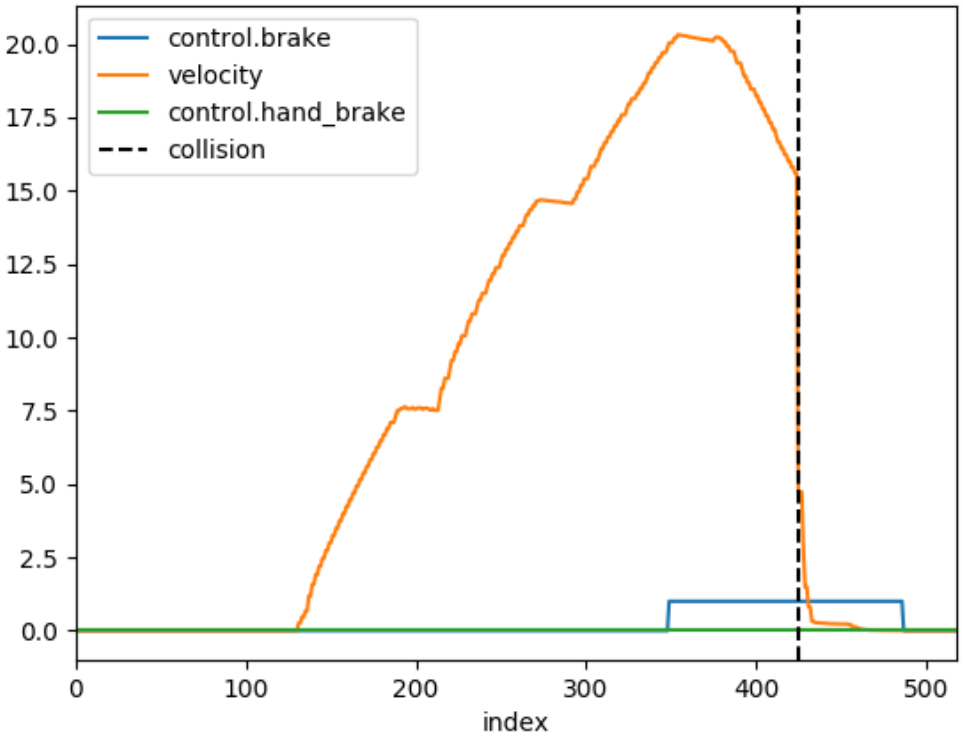


Figure 5.3: No anomaly detection run chart

From the values presented in the chart one it is immediately observed that a collision has occurred. It is seen that the vehicle began to accelerate until $x \in [370, 390]$, when the brakes control messages were sent. These however did not cause any effect on the vehicle's operation as all of them were failing and thus the vehicle relied only on the tires friction to slow down. This however was not sufficient to prevent a collision against the object present at the end of the road.

We conclude that the normal operation of a vehicle within Carla simulator is not prepared to handle brake anomalies due to the lack of a built-in anomaly detection component. To understand the efficiency of our proposed anomaly detection component, the next experiment will present its effects on the same vehicle and under the same conditions. Row (a) of table 5.4 presents a series of screenshots from this simulation, showing the collision event.

5.3.2 Attack-Free Anomaly Detection Run

This situation represents a state where the anomaly detection model is present and well-functioning as well as no attack is made to the system. Figure 5.4 presents a chart combining the values of the relevant features, `velocity`, `control.brake`, `control.hand_brake` and `PREDICTION`, for this run.

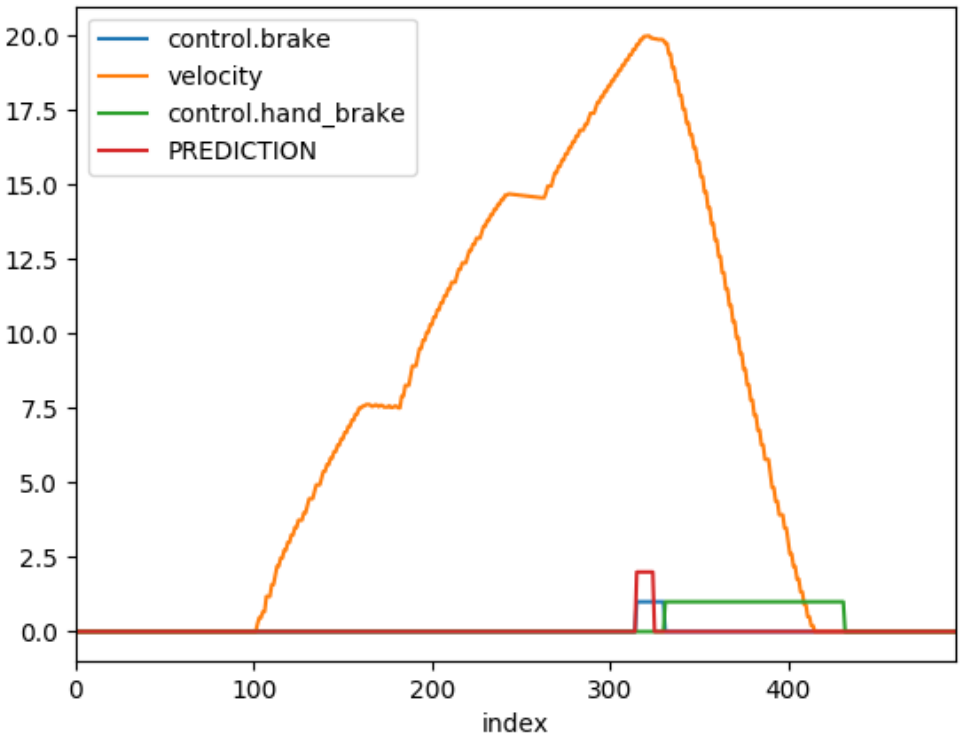


Figure 5.4: Attack-free anomaly detection run chart.

For this simulation, as opposite to the no anomaly detection run, there is no collision detected, therefore no accident happened. A careful inspection of the chart values shows that the anomaly detection model behaved correctly by first predicting the braking samples as anomalous in $x \in [310, 320]$ (`PREDICTION` = 2) during 10 consecutive anomalous samples which caused the threshold value to be crossed (originally set to ten (10)) and consequently the component's emergency mode to be activated. This resulted in emergency brake control messages to be sent to the vehicle, causing the hand brake to be activated, and thus contribute to its rapidly decrease of velocity until it finally reached a "safe stop" state (`velocity` = 0). Upon this, the emergency mode was deactivated causing a deactivation of the hand brake on the vehicle as well.

We conclude that the use of the anomaly detection component has prevented the occurrence of a collision, proving its major importance in securing autonomous vehicles and its users. Row

(b) of table 5.4 presents the series of screenshots from this simulation, showing that no collision event has happened.

5.3.3 Adversarial Run

This simulation focused on mimicking an attacker's behavior which consist on a simple man-in-the-middle attack whose goal is to inject adversarial learning generated data samples between the communication of the vehicle's sensor and the anomaly detection component in order to generate incorrect predictions. Such samples were only injected whenever a brake action was occurring, to only fool the model upon anomalous braking events so that these become undetected.

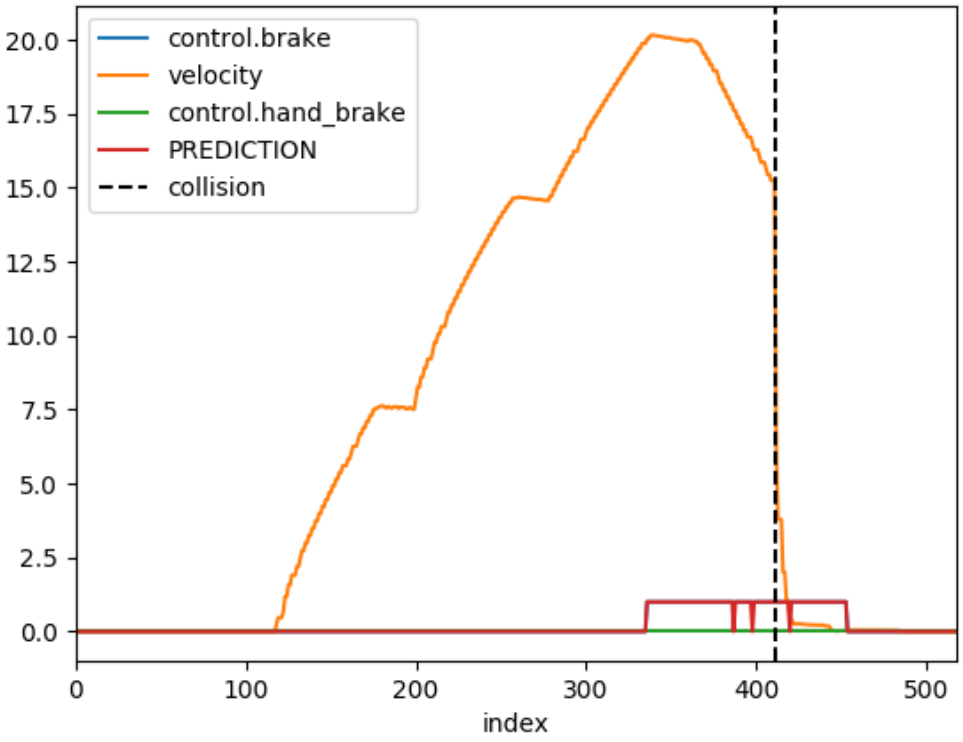


Figure 5.5: Adversarial run chart

From the values presented in the chart it can be concluded that every time the brakes were activated the model classified the injected adversarial learning samples as benign brakes (PREDICTION = 1) instead of anomalous (PREDICTION = 2). Because of this the anomaly detection component's emergency mode was not set causing no emergency brake control messages to be sent to the vehicle and consequently the hand brake was not activated. This caused the same outcome as the one for the no anomaly detection simulation, a collision.

We conclude that adversarial learning is a serious threat against modern anomaly detection models, more specifically, against the proposed brake-specific anomaly detection model and that

it is imperative to improve its robustness in order to prevent such disastrous outcomes as the one presented. Row (c) of table 5.4 presents the series of screenshots from this simulation, unveiling a similar behavior to the first simulation run as both resulted in collisions.

An additional testing was conducted by running five different simulations with the same configurations as production testing, presented in section 4.1.3, in order to assess the model's performance upon adversarial learning generated samples regarding the level of severity of the vehicle's brake anomalies. These results are presented in B.5 and reveal that the model was unable to detect any of such samples as anomalous. This reasserts the idea that the development of machine learning models, in specific, anomaly detection models has to consider such attacking techniques in order to create an efficient, robust model capable of detecting such behaviors.

5.4 Defenses

Defending machine learning models against such dangerous attacks must be of the highest priority when starting their creation process. Although not implemented, it should be noted that defense strategies such as adversarial training would improve the model's robustness against these type of attacks. To implement such strategies one should attack its own machine learning model to generate adversarial samples and inject these in the training data set so that the risk of exploit of such attacks is minimal.

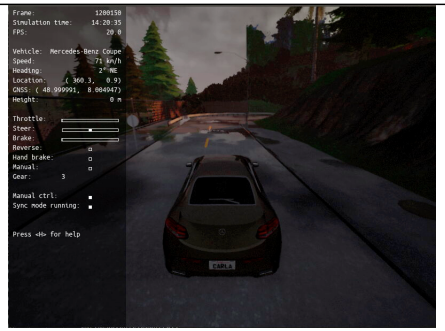
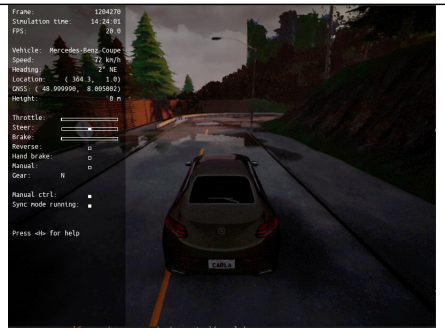
	Start	Brakes Activation	Collision	Full Stop
(a)				
	Start	**Brakes Activation**	**Emergency Brake Activation**	**Full Stop**
(b)				
	Start	**Brakes Activation**	**Adversarial Attack & Collision**	**Full Stop**
(c)				

Table 5.4: Screenshots of the different experiments ran to asses the impact of both having a normal and exploited anomaly detection model.

Chapter 6

Conclusions & Future Work

6.1 Overview

The developed work achieved its initial expectations by presenting and demonstrating a thorough and in-depth study on how adversarial learning techniques against autonomous vehicles can easily result in dangerous situations. Having the opportunity to work with the hyper-realistic Carla simulator to conduct the presented work, offered a realistic environment to extract information and later test the created brake-specific anomaly detection model in a simulated autonomous vehicle. By doing so, there is a higher level of confidence that the results presented in this work are closely applicable to those of a real-world physical autonomous vehicle.

We presented a novel anomaly detection component fully-integrated with Carla-ROS bridge permitting its users to effortlessly integrate machine learning-based anomaly detection models within Carla simulations and thus provide the means to a simpler simulation environment targeting the study of anomaly detection in autonomous driving. This component's architecture and internal workflows were described in detail, explaining its operation and features.

An extensive study of the development of a machine learning model capable of detecting brake-specific anomalies was later described presenting distinct approaches focused on one-class, binary and multi-class classification using novelty detection as well as deep learning techniques for implementation of the studied models. Different concepts and techniques were applied during both the collection and training process of the models as well as a novel testing procedure, referred to as production testing, all targeting the deliverance the most appropriate developed model for the current problem. An in-depth comparative analysis between the different approaches models led to the selection of the multi-class classification model as the most appropriate for integrating the anomaly detection component, outperforming the remaining ones in both training and testing phases.

Finally, the selected brake-specific anomaly detection model was subject to different untargeted white-box and black-box adversarial learning attacks revealing its vulnerabilities and weaknesses against such techniques. A ROS-specific attack was presented demonstrating how it can be used to exploit the normal operation of the proposed anomaly detection component. Additional

simulations were conducted focused on demonstrating the importance of a brake-specific anomaly detection model in total brake-failure situations, as well as how such mechanism might become obsolete if not robust against adversarial learning attacks, ultimately resulting in an accident.

The results presented in this work allow to understand that adversarial learning is a serious issue that should always be considered when using machine learning applications and that defenses strategies must not be forgotten as disastrous outcomes may result from its disregard.

6.2 Future Work

As future work, it would be interesting to collect more brake anomalies information so that, for supervised learning techniques, the different target classes distribution is less imbalanced and thus contribute to a more robust learning process. State-of-the-art hyperparameter optimization techniques should be studied to compare their training results against the ones from the manual tuning process.

With regard to adversarial learning, the list of techniques submitted against the created model could be expanded in order to study which attacks are more or less successful against it. On the other hand, techniques for improving the model's security against such attacks, for instance adversarial training, should be studied as well.

The techniques presented in this work can be expanded to other type of anomalies possibly becoming target classes of the developed multi-class classification model so that it is able to distinguish more occurring anomalies besides brake-specific. Ultimately, such model could be tested in a physical autonomous vehicle to assess its performance.

Appendix A

Carla-ROS Bridge Anomaly Detection Component Usage

We present here the installation steps for the successful installation of the novel Carla-ROS bridge anomaly detection component. Note that this installation guide was successfully tested with the requirements presented in table A.1.

Table A.1: Requirements for the successful installation of the anomaly detection component.

Requirement	Version
Operating System	Ubuntu 18.04
Carla Simulator	0.9.9
ROS	Melodic
Carla-ROS Bridge	0.9.8
python3	3.6.9
python	2.1.17

A.1 Installation

```
## 1. Create workspace directories.

mkdir -p ~/anomaly_detector_ws/src
cd anomaly_detector_ws/src

## 2. Pull repository.

git clone https://github.com/francismaria/carla_anomaly_detector

## 3. Chain the ROS-Bridge worskpace so that Carla messages are
    successfully built into our current workspace.

```

```
12 # ( Assuming that you already have workspace of ROS Bridge, do the
       following:)
13
14 cd ../
15 source ~/<your-path-to-Carla-ROS-Bridge>/carla-ros-bridge/catkin_ws/devel/
       setup.bash
16
17 ## 4. Build package.
18
19 catkin_make
20
21 ## 5. Source the Anomaly Detector Worskpace for running the package.
22
23 source ~/anomaly_detector_ws/devel/setup.bash
```

Listing A.1: Installation steps for the *carla_anomaly_detection* ROS package.

A.2 Usage

```
1 ## 1. Run Carla-ROS Bridge
2
3 $ export PYTHONPATH=/<your-path-to-Carla>/PythonAPI/carla/dist/carla-0.9.9-
      py2.7-linux-x86_64.egg:$PYTHONPATH &&
4 source ~/<your-path-to-Carla-ROS-Bridge>/carla-ros-bridge/catkin_ws/devel/
      setup.bash &&
5 roslaunch carla_ros_bridge carla_ros_bridge_with_example_ego_vehicle.launch
6 synchronous_mode:=true
7 synchronous_mode_wait_for_vehicle_control_command:=false
8 fixed_delta_seconds:=0.05
9 vehicle_filter:=vehicle.mercedes-benz.coupe
10
11 ## 2. Run Preprocessing Node
12
13 $ python preprocessing_node.py
14
15 ## 3. Run Anomaly Detector Node
16
17 $ python anomaly_detector_node.py
```

Listing A.2: Steps for running the *carla_anomaly_detection* ROS component.

Appendix B

Brake Anomaly Detection Model Auxiliary Information

B.1 Binary Classification Training Metrics

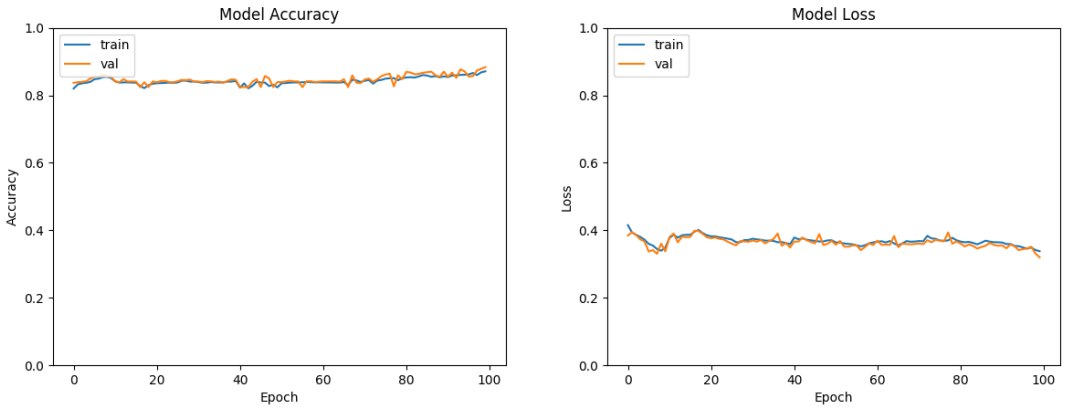


Figure B.1: Binary classification model training metrics.

B.2 Multi-Class Classification Training Metrics

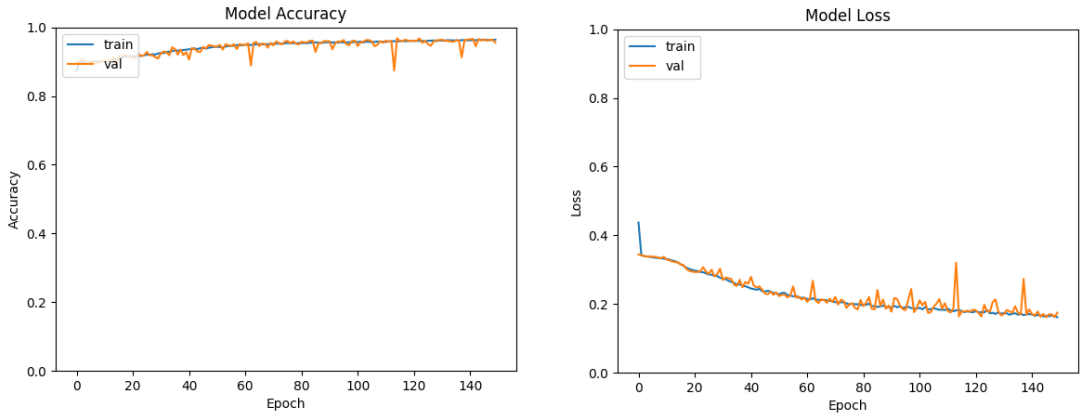


Figure B.2: Multi-class classification model training metrics.

B.3 Features Exploratory Analysis

Table B.1: Features available in the data set of the collected data.

Features	Type	Count
header.seq_x	int64	299327
header.stamp.secs	int64	299327
header.stamp.nsecs	int64	299327
header.frame_id	object	299327
velocity	float64	299327
acceleration.linear.x	float64	299327
acceleration.linear.y	float64	299327
acceleration.linear.z	float64	299327
acceleration.angular.x	float64	299327
acceleration.angular.y	float64	299327
acceleration.angular.z	float64	299327
orientation.x	float64	299327
orientation.y	float64	299327
orientation.z	float64	299327
orientation.w	float64	299327
control.header_seq	int64	299327
control.header.stamp.secs	int64	299327
control.header.stamp.nsecs	int64	299327
control.header.frame_id	float64	299327
control.throttle	float64	299327
control.steer	float64	299327
control.brake	float64	299327
control.hand_brake	bool	299327
control.reverse	bool	299327
control.gear	int64	299327
control.manual_gear_shift	bool	299327
header.seq_y	int64	299327
child.frame_id	object	299327
pose.pose.position.x	float64	299327
pose.pose.position.y	float64	299327
pose.pose.position.z	float64	299327
pose.pose.orientation.x	float64	299327
pose.pose.orientation.y	float64	299327
pose.pose.orientation.z	float64	299327
pose.pose.orientation.w	float64	299327
pose.covariance	object	299327
twist.twist.linear.x	float64	299327
twist.twist.linear.y	float64	299327
twist.twist.linear.z	float64	299327
twist.twist.angular.x	float64	299327
twist.twist.angular.y	float64	299327
twist.twist.angular.z	float64	299327
twist.covariance	object	299327

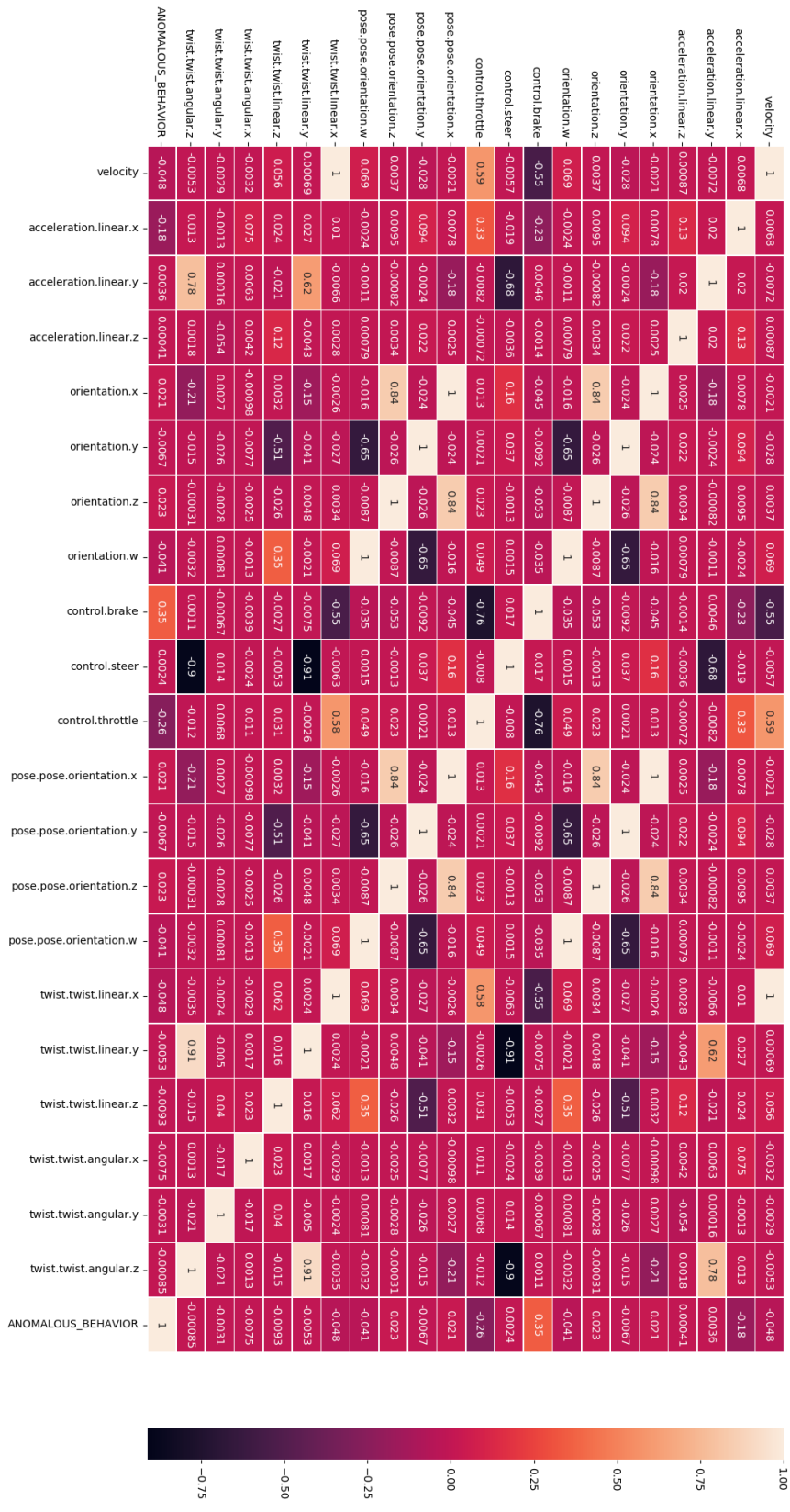


Figure B.3: Correlation matrix of the binary classification model training data set.

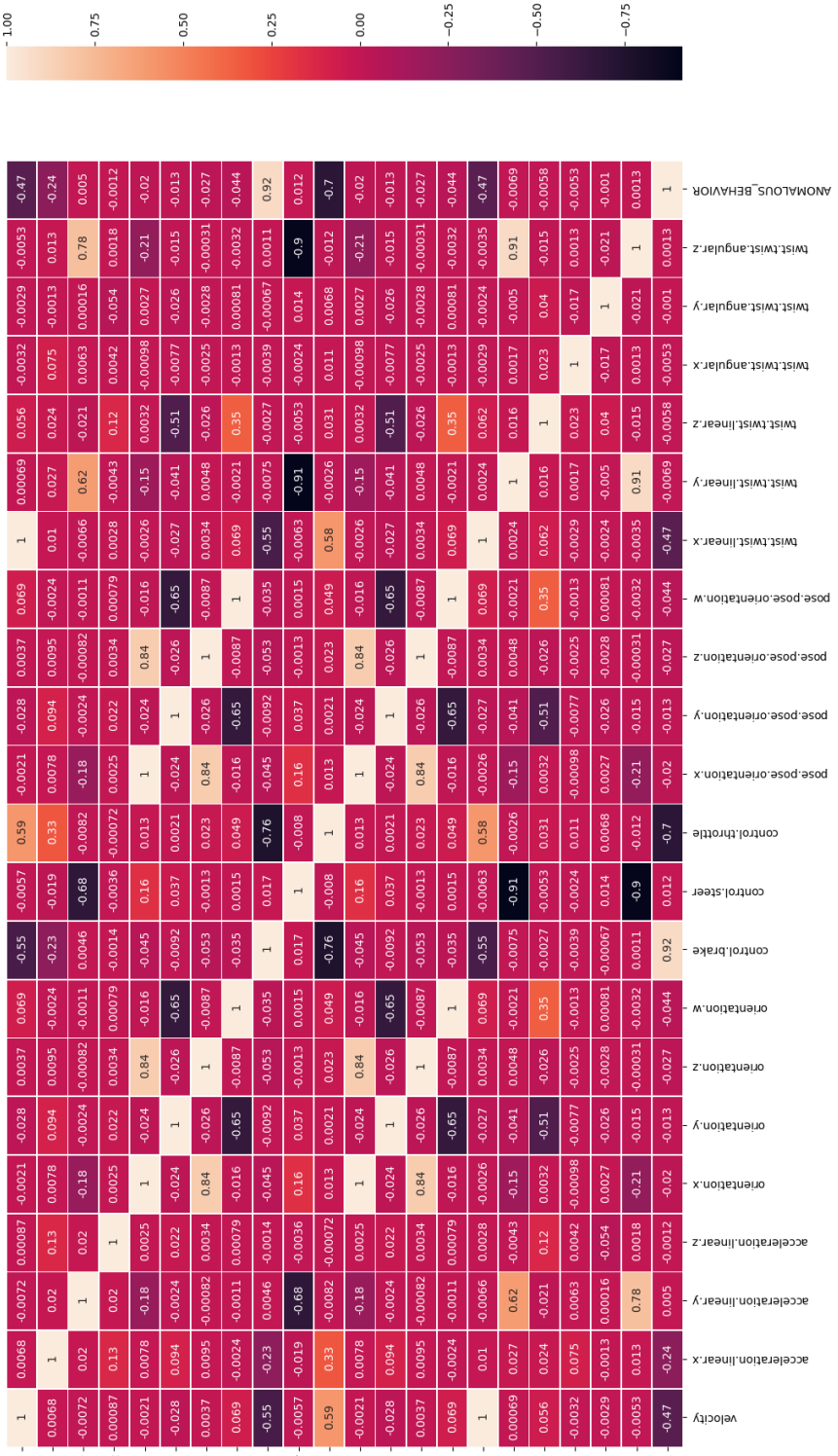


Figure B.4: Correlation matrix of the multi-class classification model training data set.

B.4 Attack-Free Anomaly Detection Production Testing Results

Table B.2: Results of the anomaly detection component predictions upon normal samples.

	Not Braking				Normal Brake				Anomalous Brake			
	PPV	TPR	F_1	ACC	PPV	TPR	F_1	ACC	PPV	TPR	F_1	ACC
#1	98,17%	100%	99,07%	98,41%	100%	31,91%	48,39%	89,84%	-	-	-	-
#2	98,95%	100%	98,47%	98,97%	-	-	-	-	100%	66,67%	80%	98,97%
#3	99,06%	100%	99,52%	98,08%	-	-	-	-	100%	71,43%	83,33%	99,08%
#4	99,51%	100%	99,76%	99,52%	-	-	-	-	100%	83,33%	90,91%	99,52%
#5	98,76%	100%	99,38%	98,79%	-	-	-	-	100%	62,5%	76,92%	98,79%

Table B.3: Units results of the functional anomaly detection component.

	Not Braking				Normal Brake				Anomalous Brake			
	TP	TN	FP	FN	TP	TN	FP	FN	TP	TN	FP	FN
#1	268	42	5	0	15	268	0	32	0	288	27	0
#2	470	10	5	0	0	485	0	0	10	470	0	5
#3	421	10	4	0	0	435	0	0	10	421	0	4
#4	409	10	2	0	0	421	0	0	10	409	0	2
#5	479	10	6	0	0	495	0	0	10	479	0	6

B.5 Attacked Anomaly Detection Production Testing Results

Table B.4: Results of the anomaly detection component prediction upon adversarial samples.

	Not Braking				Normal Brake				Anomalous Brake			
	PPV	TPR	F_1	ACC	PPV	TPR	F_1	ACC	PPV	TPR	F_1	ACC
#1	98,21%	99,40%	98,80%	98,26%	98,39%	95,31%	96,83%	98,26%	-	-	-	-
#2	97,81%	100%	98,89%	98,53%	-	-	-	-	0%	0%	0%	65,52%
#3	98,30%	100%	99,14%	98,79%	-	-	-	-	0%	0%	0%	69,64%
#4	99,28%	100%	99,64%	99,50%	-	-	-	-	0%	0%	0%	68,83%
#5	98,85%	100%	99,42%	99,21%	-	-	-	-	0%	0%	0%	67,97%

Table B.5: Units results of the exploited anomaly detection component.

	Not Braking				Normal Brake				Anomalous Brake			
	TP	TN	FP	FN	TP	TN	FP	FN	TP	TN	FP	FN
#1	329	122	6	2	122	329	2	6	0	459	0	0
#2	268	135	6	0	0	274	135	0	0	268	0	141
#3	289	121	5	0	0	294	121	0	0	289	0	126
#4	276	123	2	0	0	278	123	0	0	276	0	125
#5	259	119	3	0	0	262	119	0	0	259	0	122

Appendix C

Scripts

C.1 Parsing Rosbag (.bag) files

```
# Francisco Maria, 2019
import rosbag
import csv
import argparse

'''

This script parses .bag files to .csv.

It uses a recursive approach to loop through all the topics and all of its attributes.
For each topic in the .bag file, it creates a new .csv.

'''

# --------------------------
#        LOADS FILE
# --------------------------

def loadBag():
    '''
        Loads the bag file to be parsed.
    '''

    bag = rosbag.Bag('file.bag', 'r')
    return bag

# --------------------------
#        GET TOPICS
# --------------------------

```

```
31 def getTopics(bag):
32     '''
33         Returns all the topics names.
34     '''
35 
36     return bag.get_type_and_topic_info()[1]
37 
38 # -------------------------
39 #        GET FILENAME
40 # -------------------------
41 
42 def getFilename(topic):
43     '''
44         Creates and returns a custom filename (following a template)
45         of the csv file to be generated.
46     '''
47 
48     filename = ''
49     for i in range(len(topic)-1, 0, -1):
50         if(topic[i] == '/'):
51             break
52         filename = topic[i] + filename
53 
54     return filename + '_generated.csv'
55 
56 # -------------------------
57 #      GET ATTRIBUTES
58 # -------------------------
59 
60 def getAttributes(obj, slots, parents=''):
61     '''
62         Recursive function which takes as input:
63 
64             - obj: the object to retrieve the attributes
65             - slots: a list, representing the slots of the object
66             - parents: a string representing the objects which contain the
                attributes (slots) to be analysed
67 
68         It returns a list of all the elements to be added to the .csv
69     '''
70     resAttrs = []
71 
72     for attr in slots:
73         if isinstance(getattr(obj, attr), (int, float, str, tuple, list)):
74             resAttrs.append('%s%s' % (parents, attr))
75         else:
76             parentsComposition = '%s%s.' % (parents, attr)
77             resAttrs = resAttrs + getAttributes(getattr(obj, attr), getattr
                (obj, attr).__slots__, parentsComposition)
```

```
78
79      return resAttrs
80
81  # --------------------------
82  #       GET VALUES
83  # --------------------------
84
85  def getValues(obj, slots):
86      '''
87          Recursive function which takes as input:
88
89              - parent: an object representing the parent
90
91          It returns a list of all the elements to be added to the .csv
92      '''
93      values = []
94
95      for attr in slots:
96          if isinstance(getattr(obj, attr), (int, float, str, tuple, list)):
97              values.append(getattr(obj, attr))
98          else:
99              values = values + getValues(getattr(obj, attr), getattr(obj,
                    attr).__slots__)
100
101     return values
102
103 # ----------------------------
104 #   CREATE CSV FILE FOR TOPIC
105 # ----------------------------
106
107 def createTopicCSVFile(bag, topics, topic_name):
108     msg_type = topics[topic_name].msg_type
109     topic_msgs = bag.read_messages(topics=topic_name)
110     # first message to know its type attributes (all subsequent messages
            will have these attributes)
111     msg = topic_msgs.next().message
112     attributes = getAttributes(msg, type(msg).__slots__)
113
114     with open(getFilename(topic_name), 'wb') as csvfile:
115         filewriter = csv.writer(csvfile, delimiter=',')
116         filewriter.writerow(attributes)
117
118         for single_msg in topic_msgs:
119             values = getValues(single_msg.message, type(single_msg.message)
                    .__slots__)
120             filewriter.writerow(values)
121
122     csvfile.close()
123
```

```
124 # ----------------------------
125 #             MAIN
126 # ----------------------------
127
128 def main():
129     bag = loadBag()
130     topics = getTopics(bag)
131
132     for topic_name in topics:
133         print('Generating ' + topic_name + ' .csv file...')
134         createTopicCSVFile(bag, topics, topic_name)
135         print('Generation has finished.\n\n')
136
137 if(__name__ == '__main__'):
138     main()
```

Listing C.1: Python script for parsing a .bag file

References

[1] Ahmed Abusnaina, Aminollah Khormali, Hisham Alasmary, Jeman Park, Afsah Anwar, and Aziz Mohaisen. Adversarial Learning Attacks on Graph-based IoT Malware Detection Systems. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1296–1305, Dallas, TX, USA, July 2019. IEEE.

[2] I. Alrashdi, A. Alqazzaz, E. Aloufi, R. Alharthi, M. Zohdy, and H. Ming. Ad-iot: Anomaly detection of iot cyberattacks in smart city using machine learning. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0305–0310, Jan 2019.

[3] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square Attack: a query-efficient black-box adversarial attack via random search. *arXiv:1912.00049 [cs, stat]*, November 2019. arXiv: 1912.00049.

[4] Jitendra N. Bajpai. Emerging vehicle technologies & the search for urban mobility solutions. *Urban, Planning and Transport Research*, 4(1):83–100, 2016.

[5] Richard Ernest Bellman. *Dynamic Programming*. Dover Publications, Inc., USA, 2003.

[6] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011*, pages 1–9, 2011.

[7] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. Practical Black-Box Attacks on Deep Neural Networks Using Efficient Query Mechanisms. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, volume 11216, pages 158–174. Springer International Publishing, Cham, 2018.

[8] Keshav Bimbraw. Autonomous Cars: Past, Present and Future - A Review of the Developments in the Last Century, the Present Scenario and the Expected Future of Autonomous Vehicle Technology:. In *Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics*, pages 191–198, Colmar, Alsace, France, 2015. SCITEPRESS - Science and and Technology Publications.

[9] A. Bounsiar and M. G. Madden. Kernels for one-class support vector machines. In *2014 International Conference on Information Science Applications (ICISA)*, pages 1–4, 2014.

[10] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models. *arXiv:1712.04248 [cs, stat]*, February 2018. arXiv: 1712.04248.

[11] Yulong Cao, Chaowei Xiao, Benjamin Cyr, Yimeng Zhou, Won Park, Sara Rampazzi, Qi Alfred Chen, Kevin Fu, and Z. Morley Mao. Adversarial Sensor Attack on LiDAR-based Perception in Autonomous Driving. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security - CCS '19*, pages 2267–2281, 2019. arXiv: 1907.06826.

[12] Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks. *Proceedings - IEEE Symposium on Security and Privacy*, pages 39–57, 2017.

[13] Raghavendra Chalapathy and Sanjay Chawla. Deep Learning for Anomaly Detection: A Survey. *arXiv:1901.03407 [cs, stat]*, January 2019. arXiv: 1901.03407.

[14] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002. arXiv: 1106.1813.

[15] Nicolas Cheifetz, Allou Same, Patrice Aknin, and Emmanuel Verdalle. A pattern recognition approach for anomaly detection on buses brake system. pages 266 –271, 10 2011.

[16] Jianbo Chen, Michael I. Jordan, and Martin J. Wainwright. HopSkipJumpAttack: A Query-Efficient Decision-Based Attack. *arXiv:1904.02144 [cs, math, stat]*, September 2019. arXiv: 1904.02144.

[17] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. ZOO: Zeroth Order Optimization based Black-box Attacks to Deep Neural Networks without Training Substitute Models. *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security - AISec '17*, pages 15–26, 2017. arXiv: 1708.03999.

[18] Minhao Cheng, Thong Le, Pin-Yu Chen, Jinfeng Yi, Huan Zhang, and Cho-Jui Hsieh. Query-Efficient Hard-label Black-box Attack:An Optimization-based Approach. *arXiv:1807.04457 [cs, stat]*, July 2018. arXiv: 1807.04457.

[19] Shuyu Cheng, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. Improving Black-box Adversarial Attacks with a Transfer-based Prior. *arXiv:1906.06919 [cs, stat]*, October 2019. arXiv: 1906.06919.

[20] Kyong Tak Cho, Kang G. Shin, and Taejoon Park. CPS approach to checking norm operation of a brake-By-Wire system. *ACM/IEEE 6th International Conference on Cyber-Physical Systems, ICCPS 2015*, pages 41–50, 2015.

[21] Bernhard Dieber, Benjamin Breiling, Sebastian Taurer, Severin Kacianka, Stefan Rass, and Peter Schartner. Security for the Robot Operating System. *Robotics and Autonomous Systems*, 98:192–203, December 2017.

[22] Bernhard Dieber, Ruffin White, Sebastian Taurer, Benjamin Breiling, Gianluca Caiazza, Henrik Christensen, and Agostino Cortesi. *Penetration Testing ROS*, pages 183–225. Springer International Publishing, Cham, 2020.

[23] Yinpeng Dong, Hang Su, Baoyuan Wu, Zhifeng Li, Wei Liu, Tong Zhang, and Jun Zhu. Efficient Decision-Based Black-Box Adversarial Attacks on Face Recognition. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7706–7714, Long Beach, CA, USA, June 2019. IEEE.

[24] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[25] N. Duffield, P. Haffner, B. Krishnamurthy, and H. Ringberg. Rule-Based Anomaly Detection on IP Flows. In *IEEE INFOCOM 2009 - The 28th Conference on Computer Communications*, pages 424–432, Rio De Janeiro, Brazil, April 2009. IEEE.

[26] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. HotFlip: White-Box Adversarial Examples for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 31–36, Melbourne, Australia, 2018. Association for Computational Linguistics.

[27] David Elliott, Walter Keen, and Lei Miao. Recent advances in connected and automated vehicles. *Journal of Traffic and Transportation Engineering (English Edition)*, 6(2):109–131, April 2019.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[29] Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. Making machine learning robust against adversarial inputs. *Communications of the ACM*, 61(7):56–66, June 2018.

[30] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv:1412.6572 [cs, stat]*, March 2015. arXiv: 1412.6572.

[31] Ritwik Gupta, Zachary T. Kurtz, Sebastian Scherer, and Jonathon M. Smereka. Open Problems in Robotic Anomaly Detection. *arXiv:1809.03565 [cs]*, September 2018. arXiv: 1809.03565.

[32] Kyoung Soo Han, Jae Hyun Lim, Boojoong Kang, and Eul Gyu Im. Malware analysis using visualized images and entropy graphs. *International Journal of Information Security*, 14(1):1–14, February 2015.

[33] Haibo He, Yang Bai, Edwardo Garcia, and Shutao Li. ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1322 – 1328, 2008.

[34] Zhichao Huang and Tong Zhang. Black-Box Adversarial Attack with Transferable Model-based Embedding. *arXiv:1911.07140 [cs, stat]*, January 2020. arXiv: 1911.07140.

[35] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[36] Aminollah Khormali, Ahmed Abusnaina, Songqing Chen, DaeHun Nyang, and Aziz Mohaisen. COPYCAT: Practical Adversarial Attacks on Visualization-Based Malware Detection. *arXiv:1909.09735 [cs]*, September 2019. arXiv: 1909.09735.

[37] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial Machine Learning at Scale. *arXiv:1611.01236 [cs, stat]*, February 2017. arXiv: 1611.01236.

[38] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings*, (c):1–14, 2019.

[39] Sofiane Lagraa, Maxime Cailac, Sean Rivera, Frederic Beck, and Radu State. Real-Time Attack Detection on Robot Cameras: A Self-Driving Car Application. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 102–109, Naples, Italy, February 2019. IEEE.

[40] Huiying Li, Shawn Shan, Emily Wenger, Jiayun Zhang, Haitao Zheng, and Ben Y. Zhao. Blacklight: Defending Black-Box Adversarial Attacks on Deep Neural Networks. *arXiv*, 2020.

[41] Zudi Lin, Hanspeter Pfister, and Ziming Zhang. White-Box Adversarial Defense via Self-Supervised Data Estimation. *arXiv:1909.06271 [cs, stat]*, September 2019. arXiv: 1909.06271.

[42] Xinbo Liu, Jiliang Zhang, Yaping Lin, and He Li. ATMPA: Attacking Machine Learning-based Malware Visualization Detection Methods via Adversarial Examples. *arXiv:1808.01546 [cs]*, December 2019. arXiv: 1808.01546.

[43] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into Transferable Adversarial Examples and Black-box Attacks. *arXiv:1611.02770 [cs]*, February 2017. arXiv: 1611.02770.

[44] P Marín, A Hussein, D Martín, F García, De Escalera A, J M Armingol, and Carlos Iii. ROS-based architecture for autonomous vehicles. *SEGVAUTO-TRIES-CM: Technologies for a Safe, Accessible and Sustainable Mobility*, (November):1–4, 2016.

[45] Shigeru Maya, Ken Ueno, and Takeichiro Nishikawa. dLSTM: a new approach for anomaly detection using deep learning with delayed prediction. *International Journal of Data Science and Analytics*, 8(2):137–164, September 2019.

[46] Dr Miraz, Maaruf Ali, Peter Excell, and Rich Picking. A review on Internet of Things (IoT), Internet of Everything (IoE) and Internet of Nano Things (IoNT). pages 219–224, 2015.

[47] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. DeepFool: a simple and accurate method to fool deep neural networks. *arXiv:1511.04599 [cs]*, July 2016. arXiv: 1511.04599.

[48] Nir Morgulis, Alexander Kreines, Shachar Mendelowitz, and Yuval Weisglass. Fooling a Real Car with Adversarial Traffic Signs. *arXiv:1907.00374 [cs, eess]*, June 2019. arXiv: 1907.00374.

[49] Vedanth Narayanan and Rakesh B. Bobba. Learning Based Anomaly Detection for Industrial Arm Applications. In *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy - CPS-SPC '18*, pages 13–23, Toronto, Canada, 2018. ACM Press.

[50] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. Malware images: visualization and automatic classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security - VizSec '11*, pages 1–7, Pittsburgh, Pennsylvania, 2011. ACM Press.

[51] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. Adversarial robustness toolbox v1.2.0. *CoRR*, 1807.01069, 2018.

[52] H. Ning, C.-L Nan, L. Yang, X.-M Zhao, H.-X Liu, and D. Zhou. Anomaly detection of automobile braking curves. *Jiaotong Yunshu Gongcheng Xuebao/Journal of Traffic and Transportation Engineering*, 18:82–92, 12 2018.

[53] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. *arXiv:1605.07277 [cs]*, May 2016. arXiv: 1605.07277.

[54] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples. page 16.

[55] Lúcio Sanchez Passos and Rosaldo J F Rossetti. Intelligent Transportation Systems: a ubiquitous perspective. page 12.

[56] Sharon L. Poczter and Luka M. Jankovic. The Google Car: Driving Toward A Better Future? *Journal of Business Case Studies (JBCS)*, 10(1):7, December 2013.

[57] Adnan Qayyum, Muhammad Usama, Junaid Qadir, and Ala Al-Fuqaha. Securing Connected Autonomous Vehicles: Challenges Posed by Adversarial Machine Learning and the Way Forward. *IEEE Communications Surveys and Tutorials*, 22(2):998–1026, 2020.

[58] Sean Rivera, Sofiane Lagraa, Cristina Nita-Rotaru, Sheila Becker, and Radu State. ROS-Defender: SDN-Based Security Policy Enforcement for Robotic Applications. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 114–119, San Francisco, CA, USA, May 2019. IEEE.

[59] Sean Rivera, Sofiane Lagraa, and Radu State. ROSploit: Cybersecurity Tool for ROS. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 415–416, Naples, Italy, February 2019. IEEE.

[60] Benjamin I.P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J. D. Tygar. ANTIDOTE: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference - IMC '09*, page 1, Chicago, Illinois, USA, 2009. ACM Press.

[61] Yalin E. Sagduyu, Yi Shi, and Tugba Erpek. IoT Network Security from the Perspective of Adversarial Deep Learning. *arXiv:1906.00076 [cs]*, May 2019. arXiv: 1906.00076.

[62] Tara Salman, Deval Bhamare, Aiman Erbad, Raj Jain, and Mohammed Samaka. Machine Learning for Anomaly Detection and Categorization in Multi-cloud Environments. *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 97–103, June 2017. arXiv: 1812.05443.

[63] Bernhard Scholkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the Support of a High-Dimensional Distribution. *Neural Computation*, 13:1443–1471, 2001.

[64] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *arXiv preprint arXiv:1904.12843*, 2019.

[65] P. Sharma, D. Austin, and H. Liu. Attacks on machine learning: Adversarial examples in connected and autonomous vehicles. In *2019 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–7, 2019.

[66] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. Certified Defenses for Data Poisoning Attacks. page 13.

[67] Andreas Theissler. Anomaly detection in recordings from in-vehicle networks. 09 2014.

[68] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The Space of Transferable Adversarial Examples. *arXiv:1704.03453 [cs, stat]*, May 2017. arXiv: 1704.03453.

[69] Unknown. rospy_tutorials/Tutorials/numpy. Available at http://wiki.ros.org/rospy_tutorials/Tutorials/numpy, Accessed last time in 1st December 2020.

[70] Unknown. ROS/TCPROS. Available at http://wiki.ros.org/ROS/TCPROS, Accessed last time in 1st December 2020.

[71] Unknown. ROS 2 Documentation. Available at https://index.ros.org/doc/ros2/, Accessed last time in 9th February 2020, 2008.

[72] Franco van Wyk, Yiyang Wang, Anahita Khojandi, and Neda Masoud. Real-Time Sensor Anomaly Detection and Identification in Automated Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–13, 2019.

[73] Xiaoyang Wang, Ioannis Mavromatis, Andrea Tassi, Raul Santos-Rodriguez, and Robert J. Piechocki. Location Anomalies Detection for Connected and Autonomous Vehicles. *arXiv:1907.00811 [cs, eess, stat]*, July 2019. arXiv: 1907.00811.

[74] Yizhen Wang, Somesh Jha, and Kamalika Chaudhuri. An Investigation of Data Poisoning Defenses for Online Learning. *arXiv:1905.12121 [cs, stat]*, October 2019. arXiv: 1905.12121.

[75] Ill-Young Weon, Doo Song, Sung-Bum Ko, and Chang-Hoon Lee. A Multiple Instance Learning Problem Approach Model to Anomaly Network Intrusion Detection. *JIPS*, 1:14–21, 2005.

[76] Weng-Keen Wong, Andrew Moore, Gregory Cooper, and Michael Wagner. Rule-based anomaly pattern detection for detecting disease outbreaks. In *Eighteenth National Conference on Artificial Intelligence*, page 217–223, USA, 2002. American Association for Artificial Intelligence.

[77] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications. *arXiv*, pages 1–56, 2020.

[78] Wei Emma Zhang, Quan Z. Sheng, Ahoud Alhazmi, and Chenliang Li. Adversarial Attacks on Deep Learning Models in Natural Language Processing: A Survey. *arXiv:1901.06796 [cs]*, April 2019. arXiv: 1901.06796.

[79] Xuezhou Zhang, Xiaojin Zhu, and Laurent Lessard. Online Data Poisoning Attack. *arXiv:1903.01666 [cs, stat]*, May 2019. arXiv: 1903.01666.

[80] Yuchen Zhang and Percy Liang. Defending against Whitebox Adversarial Attacks via Randomized Discretization. *arXiv:1903.10586 [cs, stat]*, March 2019. arXiv: 1903.10586.

[81] Pu Zhao, Sijia Liu, Pin-Yu Chen, Nghia Hoang, Kaidi Xu, Bhavya Kailkhura, and Xue Lin. On the Design of Black-Box Adversarial Examples by Leveraging Gradient-Free Optimization and Operator Splitting Method. page 10.