The Resilience of Deep Learning Intrusion Detection Systems for Automotive Networks

Ivo Zenden

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Stockholm, Sweden 16.7.2022

Supervisor

Prof. György Dán (KTH)

Advisors

Yeongwoo Kim (KTH)

Rolf Blom (RISE)

Han Wang (RISE)

Aalto University
School of Electrical Engineering

Aalto University
School of Electrical Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Ivo Zenden

Title The Resilience of Deep Learning Intrusion Detection Systems for Automotive Networks

Degree programme ICT Innovation - EIT Digital

Major Autonomous Systems **Code of major** ELEC3055

Supervisor Prof. György Dán (KTH)

Advisors Yeongwoo Kim (KTH), Rolf Blom (RISE), Han Wang (RISE)

Date 16.7.2022 **Number of pages** 127+14 **Language** English

Abstract

This thesis will cover the topic of cyber security in vehicles. Current vehicles contain many computers which communicate over a controller area network. This network has many vulnerabilities which can be leveraged by attackers. To combat these attackers, intrusion detection systems have been implemented. The latest research has mostly focused on the use of deep learning techniques for these intrusion detection systems. However, these deep learning techniques are not foolproof and possess their own security vulnerabilities. One such vulnerability comes in the form of adversarial samples. These are attacks that are manipulated to evade detection by these intrusion detection systems. In this thesis, the aim is to show that the known vulnerabilities of deep learning techniques are also present in the current state-of-the-art intrusion detection systems.

The presence of these vulnerabilities shows that these deep learning based systems are still to immature to be deployed in actual vehicles. Since if an attacker is able to use these weaknesses to circumvent the intrusion detection system, they can still control many parts of the vehicles such as the windows, the brakes and even the engine.

Current research regarding deep learning weaknesses has mainly focused on the image recognition domain. Relatively little research has investigated the influence of these weaknesses for intrusion detection, especially on vehicle networks. To show these weaknesses, firstly two baseline deep learning intrusion detection systems were created. Additionally, two state-of-the-art systems from recent research papers were recreated. Afterwards, adversarial samples were generated using the fast gradient-sign method on one of the baseline systems. These adversarial samples were then used to show the drop in performance of all systems.

The thesis shows that the adversarial samples negatively impact the two baseline models and one state-of-the-art model. The state-of-the-art model's drop in performance goes as high as 60% in the f1-score. Additionally, some of the adversarial samples need as little as 2 bits to be changed in order to evade the intrusion detection systems.

Keywords Vehicle Security, Deep Learning, Controller Area Network, Intrusion Detection System, Adversarial Samples

Acknowledgments

I would like to thank Rolf Blom and Han Wang who acted as my supervisors at RISE for their guidance and teachings during the project. Additionally, I would like to thank Alfonso Iacovazzi who was able to provide help regarding any machine learning related questions. At KTH, I thank my examiner György Dán and my supervisor Yeongwoo Kim for answering questions and reviewing my documents. And finally, I want to thank Arash Vahidi, who acted as my original supervisor at RISE for the first part of the thesis.

Additionally, I wish to thank the authors of [1] and the publisher Elsevier for providing a copyright license for the use of their images in this thesis report.

Stockholm, July 2022
Ivo Zenden

Contents

List of Figures

List of Tables

List of Algorithms

List of acronyms and abbreviations

AE	Autoencoder
AGRU	Attention-based Gated Recurrent Units
ANN	Artificial Neural Network
BIM	Basic Iterative Method
CAN	Controller Area Network
CIDS	Clock-based Intrusion Detection System
CNN	Convolutional Neural Network
CRC	Cyclic Redundancy Check
DBN	Deep Belief Network
DL	Deep Learning
DLC	Data Length Code
DNN	Deep Neural Network
DoS	Denial of Service
DT	Decision Tree
ECU	Electronic Control Unit
FGSM	Fast-Gradient Sign Method
GAN	Generative Adversarial Network
GRU	Gated Recurrent Units
HIDS	Host Intrusion Detection System
HMM	Hidden Markov Model
IDS	Intrusion Detection System
IoT	Internet of Things
IT	Information Technology
JSMA	Jacobian-based Saliency Map Attack
KNN	K-Nearest Neighbor

LR	Logistic Regression
LSTM	Long Short-Term Memory
MAC	Message Authentication Code
ML	Machine Learning
MLP	Multi-Layer Perceptron
NIDS	Network Intrusion Detection System
OCSVM	One-Class Support Vector Machine
PCA	Principal Component Analysis
RBM	Restricted Boltzmann Machine
RF	Random Forest
RNN	Recurrent Neural Network
SVM	Support Vector Machine
VIDS	Voltage-based Intrusion Detection System

Chapter 1

Introduction

This chapter provides the initial goal and scope of the thesis. First, summarized background information will be described. This will be followed by the problem statement and the corresponding goals of this thesis. Next, a summary of the methodology used to reach these goals will be given, as well as the ethical aspects of the thesis and the methods. A description of the delimitations of the thesis will be provided, stating what will be outside of the scope of the thesis. Finally, the structure of the rest of the thesis will be shown.

1.1 Background

Modern cars use increasingly more software and electronics.The number of lines of code reaches the tens of millions, and the amount of computers, called Electronic Control Units (ECUs), lies around the 70 on average [2]. These ECUs work together to control the car by communicating over the Controller Area Network (CAN). The increase in electronics allows for more functionality and other improvements such as comfort, however it also increases the number of security vulnerabilities of cars.

These vulnerabilities result from the increase in attack surfaces [3] as well as inherent security weaknesses of ECUs and CAN [4, 5]. The specific vulnerabilities will be elaborated on later in the thesis. An attacker is able to leverage these vulnerabilities to send messages on the CAN bus and even gain full control over the ECUs [5]. This in turn enables the attacker to do everything from rolling down a window, to blocking the brakes and/or killing the engine [5].

To address these vulnerabilities, several techniques and methods have been introduced. These range from adding message authentication codes [6], to including a security module with cryptographic capabilities [7]. One of the proposed solutions is an Intrusion Detection System (IDS), which is a technique that is already widely used in standard Information Technology (IT). This technique is able to look at the behaviour and messages of a system, and detect any anomalous activity.

An IDS is an application of anomaly detection and can be implemented in various ways. Older techniques include statistical and probabilistic models. While newer techniques focus more on machine learning methods such as K-Nearest Neighbor (KNN), Support Vector Machine (SVM) and Random Forest (RF) [8]. A recent trend is the use of Deep Learning (DL) approaches such as Convolutional Neural Networks (CNNs) and Autoencoders (AEs) [9]. Since DL techniques have achieved substantial results in several fields, it makes them an attractive option for researchers in this field.

However, even DL-based IDSs are not foolproof. As always, the security field is an ongoing battle between attackers and defenders trying to out-do each other. So the attackers will try to find and utilize the weaknesses of the DL-based IDS to circumvent it and perform their intended attacks. Therefore, to improve the safety of vehicles and their passengers, it is of importance to investigate and be aware of the weaknesses of DL-based IDSs.

1.2 Problem

In standard IT networks, the presence of security measures such as IDSs is a necessary factor to ensure the correct and secure workings of such networks. In vehicle networks, these measures have an additional importance in guaranteeing the safety of the passengers as well as bystanders. Therefore, good performance of such systems in the presence of adversaries is a requirement. Currently, there are many papers describing DL-based IDS systems which achieve a stellar performance in detecting attacks on the networks. However, the number of papers which include attacks on the IDSs themselves is extremely limited. So it is unclear how these state-of-the-art IDSs will perform while being targeted by adversaries.

1.2.1 Original problem and definition

The specific research question of this thesis is as follows:
What are the weaknesses of DL techniques that are still present in the state-of-the-art DL-based IDSs for CAN? Do these weaknesses influence their

performance when leveraged by attackers?

The corresponding hypothesis is as follows:
The weaknesses of the DL techniques will be present in the state-of-the-art IDSs. The performance will decrease when these weaknesses are leveraged.

How this performance is measured will be described later in Chapter 3.

1.3 Purpose

The presence of attackers in standard IT networks provides many issues regarding the security of systems using these networks. Due to the implementation of similar networks in vehicles these issues obviously follow. However, in addition to security issues, there are also several safety issues that can occur due to attackers. In order to increase the security and safety of modern vehicles RISE, in cooperation with other companies such as Volvo, started a large project focused on ”Cyber Resilience of Vehicles”. This project focused on multiple security angles of vehicles, and this thesis is one of these angles. The purpose of this larger project, and in extension this thesis, is to increase the reliability and safety of modern vehicles. This will reduce future risks of vehicle manufacturers and help keep users of these vehicles safe.

If the goals of this thesis are reached, it will bring light to the current limitations of the DL-based IDSs. This in turn will hopefully inspire other researchers to mind these limitations when designing new IDSs. Thus setting the first step to further improving the security and safety of our modern vehicles.

1.4 Goals

As mentioned previously, the goal of this project is to show the weaknesses and limitations of current state-of-the-art DL-based IDSs for CAN. This goal has been divided into the following four sub-goals:

1. Create two baseline DL-based IDSs
2. Re-create two state-of-the-art DL-based IDSs for CAN
3. Implement an attack that leverages a weakness of DL techniques

4. Show the effect of the implemented attack on the performance of the IDSs

These goals will be visible in the following deliverables. For each IDS, a Jupyter Notebook will be created which includes the model architecture and the parameters used during training. The code for the attack will be included in the notebook of one of the baseline IDSs. Finally, a detailed description of all models, the attack and the results will be visible in this thesis report.

1.5 Research Methodology

The first step in the methodology of this thesis performs an extensive literature research. This covered various topics such as vehicle security, anomaly detection and properties of DL techniques. After the literature research is finished, an empirical approach will be taken for the practical part of the thesis. Firstly, a public dataset containing CAN messages with attacks on the network will be chosen. Afterwards, the baseline IDSs will be designed and implemented. Next, an appropriate attack method will be selected and implemented on one of the baseline models. Following this, two state-of-the-art models will be determined and recreated. Finally, the attack implementation will be tested on all models that are implemented.

1.6 Ethical aspects

The ethical aspects of this thesis revolve mainly around the issues and benefits of publicizing weaknesses in security systems. Identifying and publicizing potential weaknesses raises awareness such that solutions can be found for these weaknesses. This in turn will increase the security and in this case also safety of the system and its users. Publicizing weaknesses unfortunately also makes attackers aware of these weaknesses. This could allow the attackers to misuse the findings of this thesis. So it is imperative that the methods used during the thesis allow for the identification of weaknesses in the DL-based IDSs, without allowing further exploitation of these weaknesses in actual vehicles.

1.7 Delimitations

The goal of this thesis is purely to show the current limitations of DL-based IDSs for CAN. No methods to overcome these limitations will be tested and no new model architecture will designed. Possible improvement using known methods will be discussed, however implementing and testing these methods will be left for future research.

Additionally, all the results will be based solely on the available datasets. There will be no tests on actual vehicles due to the lack of resources. Once again, these tests will be left open for future research.

1.8 Structure of the thesis

The rest of the thesis will have the following structure. Chapter 2 presents relevant background information about ”vehicle networks and security”, ”anomaly detection”, and ”deep learning techniques and weaknesses”. Chapter 3 describes the methodology and method used during the thesis. Chapter 4 shows the execution of the steps mentioned in the methodology. Next Chapter 5 will cover the relevant results with respect to the original research question. These results and their implications will then be discussed in Chapter 6. And finally, the conclusion of the thesis and potential future work will be covered in Chapter 7.

Chapter 2

Background

This chapter will provide the necessary background information for the three areas relevant to this thesis. The first area is that of vehicles. The specifics regarding CAN and ECUs will be described. Additionally, an overview of security issues regarding modern vehicles will be given. The second area revolves around Machine Learning (ML) and DL techniques. The workings of the techniques that will be used in the thesis will be described. In addition, known weaknesses and defenses will be covered. In the third area, anomaly detection and intrusion detection will be described. A description of different applications and standard techniques will be presented. After the background of each area is given, a selection of related works will be summarized.

2.1 Vehicle Specification and Security

As mentioned previously, modern vehicles contain a large amount of electronics. While most functionality in the past was handled using mechanical methods, such as the brakes, currently there is almost no functionality that is not managed by electronics in some form. The small computers that manage these functionality are called ECUs. Current vehicles contain dozens of these ECUs, each of which manage specific parts of the vehicle, ranging from the windows, the infotainment system to the engine and the brakes.

To handle the complex functionality of vehicles, these ECUs are required to communicate with each other and work together. To facilitate this communication, several different networks are implemented in vehicles. These networks include LIN, MOST, FlexRay and CAN [10]. Each of these networks have their own applications and protocols. LIN is used to connect sensors and actuators. MOST focuses on media and other infotainment data. FlexRay is used for Drive-by-Wire functionality and the safety systems of the vehicle. Finally, CAN is used for the communication regarding the main functionality of the vehicles. This thesis will also focus on the CAN communication due to its importance with respect to the functionality of the vehicle [10] and its relative simplicity regarding its protocol. The reason the protocol has to be simplistic in comparison to standard IT networking protocols, is due to the limited computational resources available in vehicles. In Figure 2.1 a schematic image of a standard CAN and ECU network can be seen. Several ECUs are connected to CAN busses, and these busses are connected to each other via a gateway ECU.

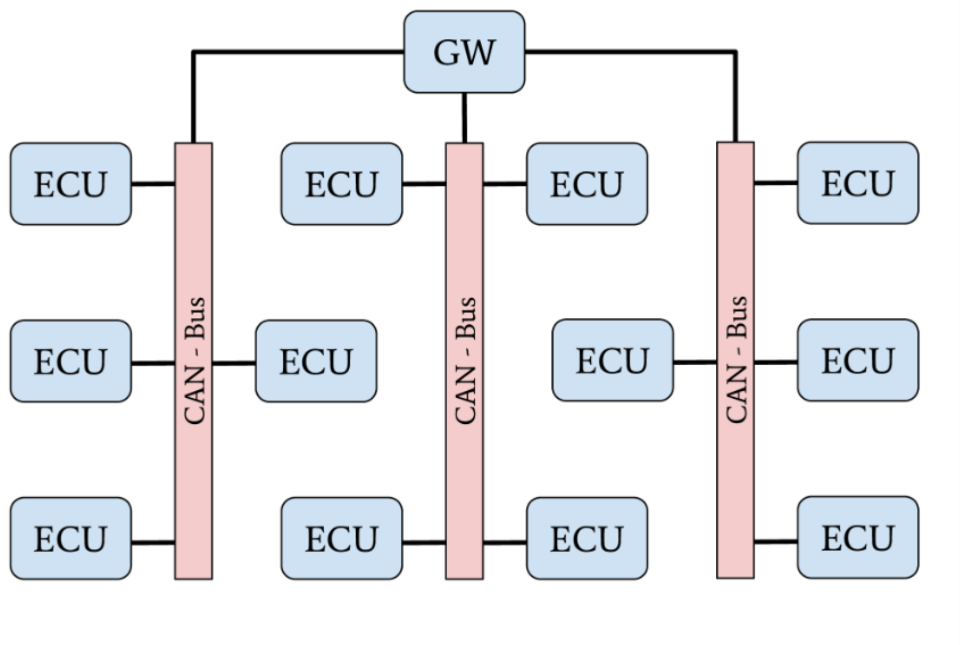


Figure 2.1: Schematic of CAN and ECU network topology [11]

2.1.1 CAN specification

In this section, a description of the CAN specification will be provided as described in [4]. Note that this description will be limited to the aspects that are relevant to this thesis. For example, while the CAN protocol defines four different types of messages: Data frame, Remote frame, Error frame and

Overload frame. Only the Data frames will be considered since they are used for the majority of the communication on the CAN bus.

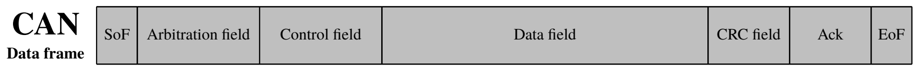


Figure 2.2: CAN data frame

A CAN data frame consists of the following fields:

- **Start of Frame (SoF)**: a bit signalling that a data frame message is starting on the CAN bus
- **Arbitration field**: a field consisting of 11 Identifier bits and 1 Remote Transmission Request (RTR) bit. The Identifier bits provide each message with a specific ID. The RTR bit is only enabled for a Remote frame.
- **Control field**: a field consisting of a 4 bit Data Length Code (DLC) and 2 reserved bits. The DLC defines how many data bytes will be present in the data field.
- **Data field**: a field consisting of 0 to 8 bytes, containing the content of the message.
- **CRC field**: a field containing a Cyclic Redundancy Check (CRC) of the message used to detect bit errors during transmission.
- **Ack field**: a field containing an Acknowledgment bit, used by receivers of messages to signal that they received the message successfully.
- **End of Frame (EoF)**: a sequence of 7 bits signalling the end of the message.

Additionally, the CAN specification also includes Extended Data frames. The only difference is that the extended frames contain a 29 bit Identifier instead of an 11 bit identifier.

As can be seen from Figure 2.2, CAN messages do not include a sender or receiver field, which are generally used in standard IT network protocols. Instead, each message has a specific ID (the Identifier bits), signalling what type of information is included in the message. The sender of the message, broadcasts its message over the CAN to all other ECUs. Then each ECU can decide whether to process or discard the message based on the message ID.

This ID also plays a role in the arbitration rule for the CAN protocol. An arbitration rule is necessary when multiple ECUs are sending their message at the same time on the network. Since if multiple messages are sent simultaneously, they will interfere with each other on the bus. The arbitration rule specifies which ECU or message will gain access to the bus when multiple messages are being sent. The arbitration rule is as follows: the lower the message ID, the higher the priority. So if two messages are being sent simultaneously, the one with the lowest ID will be allowed to continue.

2.1.2 Vehicle and CAN weaknesses

As mentioned earlier, ECUs manage various functionalities of vehicles, some of which are critical to the safety of the vehicle. As such, each ECU has a different level of risk regarding safety and security. A detailed risk analysis can be found in [12], which shows that especially the ECUs concerning the powertrain and the safety systems carry a high risk in case of attacks.

The reason that ECUs are so vulnerable to attacks comes from the limitations/simplicity of the CAN protocol [13]. First of all, CAN messages do not use any form of authentication or encryption because of the limited resources in vehicles. This means that if an attacker has access to the CAN bus, they can freely read and inject messages without having to decrypt/encrypt anything. Note that the CRC field does not prevent the injection or alteration of messages, since it can be changed to match the altered message. This allows an attacker to easily perform Fuzzy attacks, where they send messages with random IDs and data in order to find potentially dangerous combinations. Secondly, due to the lack a sender and receiver field in the messages, the authenticity of a message cannot be guaranteed. Thirdly, since all messages are being broadcasted, there are no restrictions for ECUs to send messages to other ECUs. Finally, the simplistic arbitration rule makes the CAN bus extremely vulnerable to Denial of Service (DoS) attacks. As long as many malicious messages with a low ID are sent, the actual messages (with a higher ID) will be denied from the CAN bus.

Additionally, most in-vehicle networks are connected via a gateway ECUs. Their role is to facilitate communication between ECUs on different networks by forwarding messages. Several researchers have shown that after they compromised one ECU on a network, they can use it to gain access to a gateway ECU and then to other networks [5]. This meant that the researchers only needed to compromise a single ECU, which then provided them full access to all ECUs and, as a consequence, full control of the vehicle.

Initially, these risks were manageable since getting access to that first ECU was not easily done. It usually required a longer period of time with physical access to the vehicle. However, due to all the innovations in vehicles that increased their connectivity, there are now several ways to gain access to the ECUs. Checkoway et al. [3] covered several of these attack surfaces. From indirect physical access (e.g. via the entertainment systems), to short-range wireless access (e.g. via bluetooth) and even long-range wireless access (e.g. via cellular networks). For each attack surface, several vulnerabilities and attacks were identified.

2.1.3 Vehicle security

Thus, compromising a single ECU may give an attacker access to the full vehicle, and there are several ways to gain this initial access. To secure vehicles against these types of vulnerabilities, research has been performed for both preventive and reactive measures [14]. Where preventive measures intend to stop possible attacks before they happen, usually via some form of authentication or access control methods. While reactive measures focus on detecting and then handling attacks, this is generally done using intrusion detection techniques followed by recovery actions.

For preventive measures, research has mostly focused on cryptography based solutions. Wolf et al. [10] focused on ensuring authentication by introducing public and secret key cryptography, as well as guaranteeing integrity and confidentiality by using symmetric key cryptography and firewalls. In another work, a full security module was described to protect both hardware and software using cryptography, which even enabled potential future business opportunities [7]. Finally, Nilsson et al. [6] proposed replacing the CRC fields of CAN messages with a Message Authentication Code (MAC). By authenticating four messages simultaneously and choosing resource efficient algorithms, they were able to present a solution that adheres to the resource constraints of the vehicle system. This is an important factor, since the standard cryptography algorithms used in general IT use too many resources for the automotive IT field.

As mentioned, the reactive measures aim to detect the attacks once they are happening. The main goal is to differentiate normal behaviour in the system from adversarial behaviour. This can be done by defining a set of rules which specify the expected behaviour of the system, as was done in [15]. If messages should break these defined rules, they will be classified as adversarial. In standard IT, one of the main reactive measures are IDSs. These IDSs are a well established measure, which can be implemented using various techniques (further explanation follows later). However, they still face some challenges in the automotive field [16]. One of the challenges was mentioned earlier: the limited amount of resources. This makes IDSs that use large databases with attack signatures unusable. Additionally, the consequences of having a lower detection rate and a high false positive rate are more severe. In standard IT there will be a lot of inconvenience should an attack happen, but generally no safety issues will occur. While in automotive IT, a successful attack could cause grave safety issues for the driver and their surroundings.

2.2 Machine Learning and Deep Learning

The goal of this thesis is to show the weaknesses of DL-based IDS for vehicle networks. To do this, several ML and DL techniques will be discussed and implemented. Therefore, this section will first provide a quick explanation of the techniques that are used in the later models. Note that this section will only provide a general understanding of these techniques. This means that certain details may be left out, but these details were deemed irrelevant to the thesis. Afterwards several weaknesses and defenses against these weaknesses will be described.

2.2.1 Machine Learning techniques

2.2.1.1 Logistic Regression

Logistic Regression (LR) is a statistical model that predicts the probability of an event happening based on the input variables [17]. Since the outcome is a probability, the value is limited between 0 and 1, with 0 being the event not occurring, and 1 being the event always occurring. The model can be represented by the following function:

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \tag{2.1}$$

In this equation, the betas represent the parameters that are learned/optimized by the model. Currently only one input variable is used in the formula, however it can be extended to include more variables by adding more factors to the exponent. During the fitting (training) of a LR model, generally the beta parameters are computed using the maximum likelihood estimation. This method tests various values for the beta parameters to find the values which best represent the distribution of the data.

2.2.1.2 Decision Tree

A Decision Tree (DT) is a tree-like structure that is used to classify inputs by sequentially performing decisions on the input data [18]. A DT consists of several elements: the decision nodes and the end nodes or leaves of the tree. In the decision nodes, a (binary) decision is executed on one or more of the input variables. Based on the result of this decision, the next decision node or leaf node is determined. Leaf nodes contain the final classification for the input. So a new input starts at the root of the tree, which is the initial decision node. After following a sequence of decision nodes, the input will finally reach a leaf node, which holds the corresponding classification for the input.

In a DT, the parameters that are learned, are the decisions in the decision nodes. This is done by evaluating for every decision node, what decision would result in the highest information gain. Simply put, what decision results in the cleanest split for a group of inputs with different classifications. For example, if 5 inputs arrive at the decision node. Two of which have class 0 and the other three have class 1, what decision should be made to divide these two groups.

An extension of DTs are Random Forests (RFs). For this technique, several smaller DTs are created on the original dataset. These are then combined as an ensemble method to create a single classifier for the dataset. This usually results in a better performance without exponentially increasing the computation time.

2.2.1.3 Support Vector Machine

A Support Vector Machine (SVM) aims to define the optimal decision boundary between the data points of two different classes [19]. Specifically, if all the data points are plotted, what line (decision boundary) would split these two classes of data points with the largest margin. To determine the line which provides the highest margin, support vectors are used. These are vectors from the data points closest to the line, orthogonal to the line. By optimizing the size of these support vectors, the SVM is able to find the optimal decision boundary.

Originally, a SVM was only able to act as a linear classifier. This means that the decision boundary was a linear function on the input variables. However, by applying a ”kernel trick”, a SVM is also able to act as a non-linear classifier. A kernel can be used to map the input data into a high-dimensional feature space. The goal here is to map the input data to a specific feature space in which it is linearly separable. Applying the SVM to this higher dimensional feature space, they are able to act as a non-linear classifier.

2.2.1.4 K-Nearest Neighbors

The K-Nearest Neighbors (KNN) technique, classifies new input samples based on the k closest known samples [20]. The training phase of this technique consists solely of storing the training samples. Then when classifying new samples, first the k closest neighbors to the new sample are determined. This is done using a distance or similarity measure, such as the Euclidean distance. Then the classes of these k neighbors are retrieved, and the class of the majority of the samples is then assigned to the new sample.

2.2.1.5 Naïve Bayes

The Naïve Bayes technique is a probabilistic classifier [21]. During training, the classifier determines the distributions and the probabilities of the classes and input variables. Then by looking at the probabilities retrieved from the training data, the classifier is able to compute the probability of new samples belonging to each class.

There are several variants of the Naïve Bayes classifier, depending on what method is used to describe the distribution of the input data. For example, Guassian Naïve Bayes is used for continuous input variables, and Bernoulli Naïve Bayes is used for independent binary input variables.

2.2.2 Deep Learning techniques

2.2.2.1 Neural Network

The most well known form of DL is that of Artificial Neural Networks (ANNs) also called Deep Neural Networks (DNNs) [22]. The main premise is that using artificial neurons, the computer is able to emulate the workings of the human brain. A neural network is comprised of multiple layers of neurons. These neurons perform various computations on their inputs. Each neuron has a set of weights, one for each input. The inputs are multiplied with these weights and then added together. Afterwards, this summation is then fed to an activation function. There are currently many different activation functions that are often used, such as the logistic function, hyperbolic tangent and ReLu [23]. The resulting value is then send to the neurons on the next layer, or returned as output if it is the final layer.

The training of a neural network works as follows. First, there is training dataset which has various input samples with corresponding output values. These input samples are given to the neural network, which returns a predicted output. The goal of the neural network is for this output to be as close to the actual output as possible. The actual output and the predicted output are given to a loss function, which computes a loss value based on the difference between the actual and predicted output. Afterwards, gradient descent via backwards propagation is performed on this loss value. The gradient of the loss function is computed with respect to the weights of the neurons. This means that the contribution of each weight to the loss is computed. Based on this gradient, the weights are then updated. The update amount depends on the respective gradient and the learning rate, which is a value that determines the size of the update steps. After all the weights are updated, the process is repeated. This repeats either for a certain amount of epochs, or until some stopping criteria is met.

2.2.2.2 Convolutional Neural Network

Over time, various variations on neural networks have been designed. One of these is the Convolutional Neural Network (CNN) [24]. These CNNs are mostly used for analyzing images. This is because they use several shared weights among neurons called filters. These filters look at a region of input variables (for example 3 x 3 pixels), and are moved along all the inputs. This way the input is transformed into several feature maps. These filters have several benefits. By sharing the weights, the computational requirements are reduced. And since these filters are moved over the entire input, they are ”shift-invariant”. This means that the features in the data will always be detected even if they are shifted to another location.

2.2.2.3 Recurrent Neural Network

Another variation on the neural networks are Recurrent Neural Networks (RNNs) [25]. These RNNs are suited for time sequence data, since they possess a form of internal memory. While normal neurons receive input and forward their output to the neurons on the following layer. Recurrent neurons also forward their output to themselves for the next input. This self-connection allows RNNs to take previous data points into account when handling new data points.

One downside of these self-connections, is that the resulting networks are more vulnerable to the ”vanishing gradient” problem. This is a problem that occurrs during the backpropagation process, where if the gradients become to small, no update to the weights will take place. This means that the learning capabilities of the model will stop. To combat this problem, the Long Short-Term Memory (LSTM) model was designed.

The LSTM model implements internal memory using various gates and an internal cell state. The three gates included are the input gate, output gate and the forget gate. The information regarding the previous data inputs is saved in the cell states, and the gates determine what data from the new input is added or removed from this cell state. The forget gate decides based on the input and the previous hidden state, what information should be removed from the cell state. Next, the input gate determines what new information should be included in the cell state. After the cell state has been fully updated, the output gate determines what information from the cell state should be returned as output.

2.2.3 Weaknesses

While DL algorithms have provided good results and even potential breakthroughs in several fields, their weaknesses that can be exploited should not be overlooked. These weaknesses become especially relevant in the security field, where the consequences can be quite severe. Current research has highlighted several weaknesses such as data poisoning and the possibility of reverse engineering [26], however this thesis will focus on the following two weaknesses of deep learning algorithms:

1. The susceptibility to adversarial perturbations.
2. The transferability of adversarial samples.

Both of these were first mentioned in [27], however the main focus of that paper was on the perturbations.

2.2.3.1 Perturbations

Perturbations are small changes to an input of a DL algorithm, which cause the algorithm to misclassify the input. The most well known example comes from [28], where an image classification network initially classifies the input as a panda with 57% confidence, and after adding the perturbation the input is classified as a gibbon with 99% confidence (see Figure 2.3). Additionally,

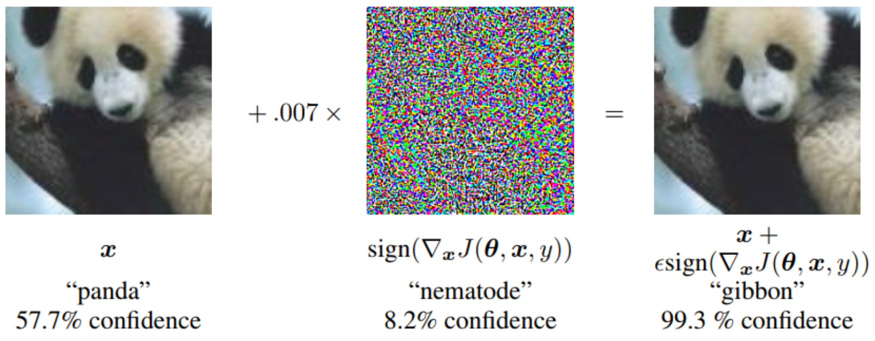

$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 2.3: Misclassification due to perturbation [28]

the perturbation is so small that the the original and the perturbed input are indistinguishable to the human eye.

Currently, many methods to compute these perturbations have been proposed, such as the Fast-Gradient Sign Method (FGSM) [28], the Jacobian-based Saliency Map Attack (JSMA) [29] and DeepFool [30]. The general idea behind most of these methods is doing the opposite of how deep learning algorithms train. When training a DL algorithm, an input is fed to the network, which then returns an output. This output results in a certain loss when compared to the desired output. Next the gradient of this loss is computed with respect to the weights of the DL algorithm. Afterwards, the weights are updated according to this gradient in order to minimize the loss. For the perturbations, the steps up to computing the loss are the same. However, after this the gradient is computed with respect to the input instead of the weights. Next the input is altered according to the gradient in order to maximize the loss. So during training, the weights of the network are updated to minimize the loss. And for perturbations the input is updated to maximize the loss. Once these perturbations are added to the input, the resulting inputs are called adversarial samples. Most research so far revolved around computing perturbations for image classification [27, 28, 30], however other fields such as cyber security [31] and malware detection have also seen research regarding adversarial samples [32]. In the rest of the thesis, the terms perturbations and adversarial samples will be used interchangeably.

2.2.3.2 Transferability

The second weakness revolves around the transferability of these adversarial samples. This entails that an adversarial sample that is generated for one deep learning model, is also capable of causing a misclassification in other deep learning models. As mentioned, the initial paper covering this weakness is that of [27], however more extensive research has been performed in [33]. This paper shows the transferability of adversarial samples not only in models using the same machine learning technique (”intra-technique”), but also between different techniques (”cross-technique”). For intra-technique, a full dataset was split into five disjoint datasets. Then a model was trained for each of the datesets. Afterwards, adversarial samples were created for each of the different models, which were then tested on the other models. This experiment was repeated for the following model types: DNN, LR, SVM, DT, KNN. The results of these experiments showed that at least 19% of the adversarial samples were misclassified when transferred intra-Technique. And especially DNN and LR models were susceptible, with respectively a minimum of 49% and 94% of misclassified samples.

For the cross-technique transferability, one model was created for each technique (DNN, LR, SVM, DT, KNN) and one ensemble model was created. Each model was then trained on the full dataset. Next, adversarial samples were generated for each model, which were then tested on the other model types. The DT was the most vulnerable to cross-technique transferability, with around 80% of the adversarial samples being misclassified. While the DNN was the most resistant, with an average of 6% being misclassified. Additionally, the ensemble method was not more resistant to the adversarial samples, with the misclassification percentages lying between 5% and 44%.

This showed a significant weakness, which enables various attacks, such as a black-box oracle attack showcased in the paper [33]. Generally, attackers will at most have oracle access to an employed model. This means that they have no knowledge regarding the model architecture, parameters and training data. However, they can provide selected inputs and receive the outputs from the model. This allows the attacker to create their own substitute dataset. After this, the attacker can create their own model trained on the substitute dataset. This resulting model can then be used as a substitute for the employed model when generating adversarial samples. Due to the transferability of adversarial samples, these samples generated on a substitute model will then also affect the employed model. The paper showcased this attack on classifiers ran on Amazon Web Services and Google Cloud Prediction, and reached misclassification percentages of 96% and 88% respectively. This shows that attackers only require oracle access to classifiers in order to effectively create adversarial samples for these classifiers.

2.2.4 Defenses

To increase the resilience of DL methods against adversarial samples, the following solutions have been proposed:

- Re-training on adversarial samples
- Defensive distillation
- Denoising techniques such as autoencoders

The first defense is relatively straightforward. The model is designed and trained as usual using the desired dataset. Afterwards, a second dataset containing adversarial samples is created by generating the adversarial samples on the initially trained model. The model is then re-trained on this adversarial dataset. By applying this technique, the model will become more accustomed to adversarial samples and the underlying decision-boundary will become robust towards such samples. In [29], training on adversarial samples reduced the success rate of such samples by 7%. Additionally, generating the new adversarial samples required a larger perturbation size for them to be successful. The networks in the DeepFool paper [30] were also made more robust by re-training using the adversarial samples generated by DeepFool. Note however, that this paper also showed an adverse affect on the robustness if the perturbations were too large. Similarly, the authors of [32] looked into testing different ratios of adversarial samples in the datasets and showed that adding adversarial samples to the training dataset does not always increase robustness. They showed that adding a limited number of adversarial samples decreased the misclassification rate. However adding more samples than a certain threshold will increase the misclassification rate. This threshold is dependent on the model architecture as well as parameters used during training.

Distillation to counteract adversarial samples was first introduced in [34]. For a standard DL classification algorithm, training is done on a dataset with inputs and classification labels as outputs. These labels are in the form of a list of mainly zeros and a single one, where the position of the one corresponds to the correct output class. Once a new input is provided to the model, it computes the probabilities of the model belonging to each of the classes. The class with the highest probability is returned as the final output of the model. In distillation, a second model is introduced which is trained on the original input dataset. However, for the outputs, the probabilities of the first model are used. So instead of the hard labels (e.g. [0, 0, 1, ..., 0]) that the first model is trained on, the distilled model is trained on the soft probabilities (e.g. [0.1, 0.1, 0.7, ..., 0.1]). Hence, making the model less likely to learn a hard decision boundary. This in turn increases the robustness of the model to adversarial samples. In the paper [34], they show the effect of distillation on two DNN model which reduces the misclassification rate from 95% to 0.5% and from 87% to 5%. There has however also been research regarding attacks that break defensive distillation [35]. They show that even distilled networks are still susceptible towards adversarial samples.

The final defense revolves around denoising or reconstructing the input data. The idea is similar to the Principal Component Analysis (PCA), where the goal is to reduce the dimensionality of the data. By compressing the data to a lower dimensionality and afterwards recomputing the original data, small changes or errors in the data can be recovered. In deep learning applications, this is generally done either with an AE or a Restricted Boltzmann Machine (RBM). These models are able to learn the distribution of the data and thus recreate their inputs. The hidden layers of these models have less neurons than the input and output layers, which forces them to encode the data to a lower dimensionality, followed by decoding them back to the original dimensionality. These techniques help against adversarial data by learning the distribution of the unperturbed data [36]. This way, once they encounter adversarial data, after encoding and decoding the data, most of the changes made by the perturbations will be removed, since they do not follow the original data distribution.

2.3 Anomaly and Intrusion Detection

Intrusion detection is an application of anomaly analysis/detection, also called outlier analysis. Outliers are data points that lie outside of the distribution defining the normal data points [37]. The presence of these outliers generally indicates something unexpected and/or undesirable is present in the system generating the data. By detecting and analyzing these outliers, the unusual behaviour can be identified and possibly mitigated. An example of data with outliers can be seen in Figure 2.4.

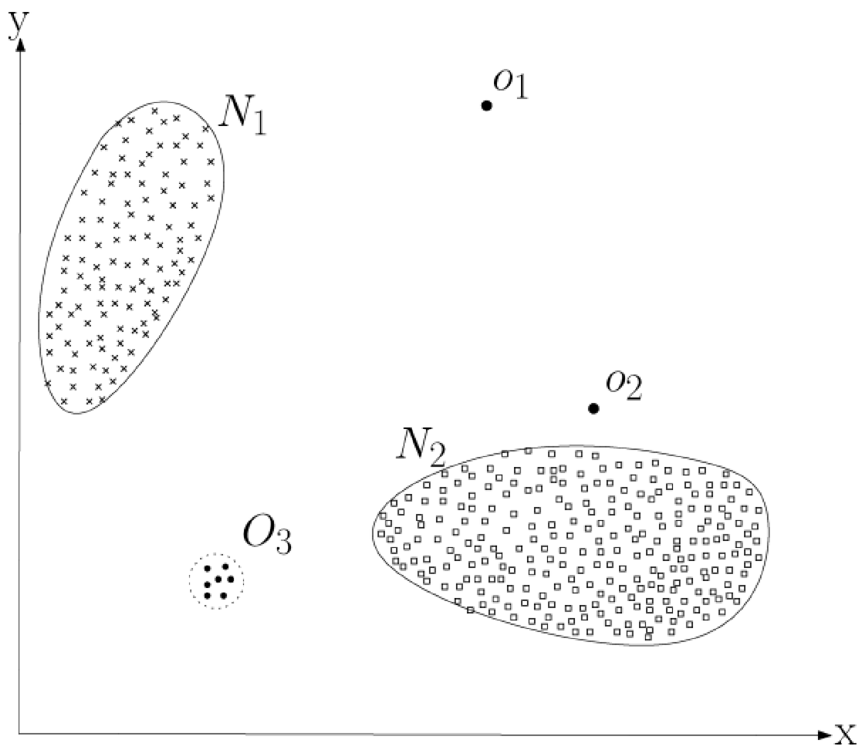


Figure 2.4: Data with outliers [38]

2.3.1 Anomaly Detection

Various techniques have been used for anomaly detection [37, 38]. Initially, anomaly detection was mainly researched with respect to the statistics field. Therefore, the most researched methods are probabilistic and statistical in nature. Most of these methods aim to define the underlying distribution of the data, and subsequently find data points that do not adhere to these distributions. Other techniques revolve around the proximity of data points to others, an example is KNN. The aim of these methods is to define data points based on their surroundings and identify anomalies by using either distance or similarity thresholds. Machine learning methods have also been extensively researched, such as DT, SVM and ensemble methods. By providing labeled data that represents the actual system, these techniques can learn how to distinguish between standard data and anomalies on their own. Following machine learning techniques, DL techniques have gained a lot of attention and have shown a similar or even better performance [9]. The ability to handle large datasets of high dimensional data, and the adaptability to different applications has resulted in a high popularity of DL techniques.

In addition to various techniques, anomaly detection also has applications in several fields from fraud detection to medical diagnosis, and also in intrusion detection for cyber security [37, 38]. The most known example for fraud detection is that of credit card fraud. Where the goal is to define the usual usage behaviour of the credit card owner, and then detect purchases that do not fit this behaviour. Medical diagnosis usually focuses on finding anomalies that signal the presence of injuries or diseases, such detecting cancer tissue using a MRI scan. In case of cyber security, the goal is to identify unusual and malicious behaviour on the system. This is generally the result of an attacker trying to make the system perform actions that are usually not allowed.

One thing to keep in mind is that simply detecting the anomaly does not resolve the reason why it occurred. For example, in the fraud detection case, either a supervisor has to be informed to block the card, or the owner has to be informed whether they performed the suspicious behaviour or not. In the medical field, the source of the anomaly, for example cancer tissue, does not disappear on its own after being detected. The doctor has to initiate the corresponding treatment. Similarly, in Cyber Security, implementing an IDS does not make the system secure or solves the actual intrusion. According to [39], which proposes an ”Anti-Intrusion Taxonomy”, intrusion detection is only one of six steps required to secure a system. These steps include: prevention, preemption, deterrence, deflection, *detection* and counter measures. Anomaly detection only shows the presence of an anomaly, it does not prevent it and it does not resolve it.

2.3.2 Intrusion Detection

Intrusion Detection Systems can be further defined in host or network IDSs, and in whether they are signature or anomaly based [8]. The distinction between host and network IDSs revolve around what part of a system the IDS is monitoring. In a Host Intrusion Detection System (HIDS), the IDS looks at internal behaviour within a system, such as file access or system commands. The goal is to prevent any unauthorized actions in the system itself. In a Network Intrusion Detection System (NIDS), the IDS monitors the network between different systems. In this situation, usually one of the systems on the network is compromised by an attacker, and the goal of the IDS is to flag any abnormal or malicious messages that appear on the network.

The way that IDSs detect intrusions can be separated into signature or anomaly detection. In signature-based detection, the IDS is trained on known attacks and forms a form of signature database for these attacks. Then when new messages appear, these messages are compared to the signatures to see if it matches any of the known attacks. Signature-based IDSs are very proficient in detecting known attacks with a low false alarm rate. However, these IDSs are generally incapable of detecting any attacks that they have not seen before. Meaning that attacks it was not trained on, or newly developed attacks will not be detected by this IDS. For anomaly-based IDSs, the IDS is trained on standard/normal behaviour of the system. These IDSs form their interpretation of normal behaviour and will then flag any messages that do not follow this interpretation. This type of IDS is capable of detecting previously unseen attacks, in contrast to signature-based IDSs. However, the downside is that previously unseen authorized behaviour is also flagged as an anomaly. This generally results in a higher false alarm rate than signature-based IDSs.

2.3.3 Network Intrusion Detection Systems

NIDSs have been applied in several IT fields already. Most research has been done regarding standard computer networks and their protocols (e.g. TCP). Because these are some of the most used networks, which means that they encounter many attacks and that numerous datasets are available. Here the increased use of machine learning and deep learning in the IDSs also applies. Generally, a deep learning technique such as AEs [40, 41, 42] or RBM [43] are used for the feature selection, afterwards more standard machine learning techniques such as SVMs and DTs are used to make the actual classification [44]. Other algorithms used include bayesian networks, clustering, ensemble methods and Hidden Markov Models (HMMs), as mentioned in [8]. The increase of Internet of Things (IoT) and wireless sensors has also prompted the application of IDSs for these fields. For example, the paper [45] proposes the use of a Deep Belief Network (DBN) using RBMs for wireless sensor networks. And the paper [46] introduces their IDS SVELTE, which uses a combination of a rule-based IDS and a specialized firewall to detect and prevent intrusions in IoT networks. The survey [47] covers several different methods for intrusion detection in IoT, including SVM, ANN and RF.

2.4 Related work

This section will cover several papers that cover topics that are relevant to this thesis. The research can be divided into two main areas. That of IDSs for vehicle networks, and attacks on these IDSs.

2.4.1 Intrusion Detection Systems for Vehicle Networks

Due to the growing number of vulnerabilities and attack surfaces of vehicles, the research regarding IDSs for in-vehicle networks has been increasing. The methods proposed vary greatly regarding techniques used, which data features are focused on, and what attacks are aimed to be detected [48, 49, 50]. For the data features, the following initial split can be determined: Hardware/physical features versus Software/digital features.

2.4.1.1 Hardware Features

Hardware features mainly focus on physical characteristics of the ECUs connected to the CAN. The goal of these IDS is to create a profile of each ECU and match them to specific CAN message IDs. This way, once a message is sent with a specific ID that does not match the profile, the IDS is able to determine that an intrusion happened. This basically allows the IDS to perform sender confirmation, which is not included in the standard CAN specification. These profiles are based one of the following physical features: clock-skew and voltage profiles.

Clock-skew is the difference of a clock's frequency with the frequency of a true clock [51]. Every ECU has their own internal clock mechanism which is used to determine when to send their next message. These internal clocks however are not perfect. Their frequency differs slightly from a true clock, meaning that they run slightly faster or slower than a true clock would. Additionally, this clock-skew is unique and constant for each ECU. This allowed the authors of [51] to introduce their Clock-based Intrusion Detection System (CIDS), which fingerprints each ECU according to their clock-skew which can then be used to detect intrusions in the system. They were able to detect various attacks such as suspension and masquerade attacks. An additional feature is that by tracking the clock-skew of the intrusion, allows CIDS to determine which ECU is compromised.

Similar to the clock-skew, each ECU has their own voltage characteristics. This originates from various factors, such as small differences in the internal transistors and voltage regulators, the length of the wire between the ECU and the measuring point and even the temperature of the vehicle, wires and ECUs. The papers for VoltageIDS [52] and VIDEN [53], propose Voltage-based

Intrusion Detection Systems (VIDSs) which use these voltage characteristics to create a profile for the ECUs. These profiles are then used to detect intrusions in a similar manner as the previously discussed CIDS. Both IDSs were able to effectively detect intrusions and additionally identify from which ECU the intrusion originated. Additionally, VIDEN introduced a ”profile adjustment” feature, which allowed the ECU to periodically update the voltage profiles for the ECUs. This is helpful since these profiles will change over time due to various factors such as battery level and temperature. And if the profiles do not match the actual ECUs, the false alarm rate will increase significantly.

2.4.1.2 Software Features

While the hardware features focused on the characteristics of the ECUs, the software features focus on the characteristics of the CAN messages themselves. The corresponding IDSs use the various features from the CAN messages, such as the ID, DLC and Datafields. Additionally, the timing between messages of the same ID are often used for attack detection. Currently proposed IDSs rarely focus on all features, since each technique used has features that are more compatible as well as features that are less compatible. This also connects to which attacks the IDS aims to detect, since these attacks all have features in which they are more apparent than others. Generally, the IDS are either more payload based, or more timing based.

Payload Based

In payload based IDSs, the focus is mostly on the datafields of the CAN messages, usually in combination with the message ID. These IDSs learn which message payloads and IDs are considered standard behaviour. Additionally, certain methods focus on the flow of the messages. Particularly, they focus on whether a sequence of messages has a logical flow or not. This allows these IDSs to detect various attacks which interrupt this flow, such as injection, fuzzing and replay attacks.

To describe standard message behaviour, the researchers of paper [54] proposed using Hamming distance to specify a ”normal” range of values. For each message ID, they computed the Hamming distance between all consecutive messages and collected the minimum and maximum value. The range defined between these values would then designate the normal behaviour, and any message that would later be detected to be outside of this range would be classified as an intrusion. In [55], a deep learning method is applied in order to learn the standard behaviour. The method in question is a

DBN, whose outputs are used by an ANN to classify intrusive messages.

IDSs that aim to detect the interruption of the flow of messages are generally RNN based. This is because RNNs posses a form of internal memory which allows them to make connections between current inputs and previous inputs. This makes them proficient for many sequence dependent applications. The paper [56] proposes and LSTM based method. For each message ID, a separate LSTM model is created. These models learn the message flow of the messages corresponding to their respective ID. These LSTMs use the payload of the current message in order to predict the message of the next message. Then depending on the difference between the predicted next message and the actual next message, the system determines whether the actual next message is an intrusion or not. Similarly, the paper [57] proposed a LSTM based IDS for attack classification. However, here the LSTM part of the model was followed by several dense layers with the aim to classify which attack was happening. So instead of explicitly predicting the next message, the model looks at the current sequence of messages and classifies it either as benign or as a specific attack type. Finally, the CANnolo IDS [58], proposes and LSTM AE IDS. The IDS takes a sequence of messages as input, and using the LSTM based AE tries to recreate this sequence. Then similar to [56] the difference is used as a base to determine whether the sequence contains an intrusion or not.

One downside of most payload based IDSs is that they require to describe the normal behaviour for each message ID separately. In [54], this required creating an Hamming range for each ID. This is not that resource intensive, however for the LSTM based IDSs, a separate IDS is required for each message ID. This can result in a decent amount of memory and computational resources necessary when employed in a vehicle. And with the limited amount of resources available, it is something to pay attention to when choosing a payload based IDS. Additionally, it is impossible to learn any connective behaviour between message IDs (e.g. a message with ID A will always be followed by a message with ID B).

Timing Based

For timing-based IDSs, the most important features are the message ID and the arrival time of the message. Generally, the IDSs either look at the time interval between messages of the same ID to see whether additional messages have been added or messages have been dropped. Or the IDS looks at the messages which arrived within a certain time frame, and determine whether an attack is included based on the number of messages of each ID within that time frame. The reason that this is a valid measure to detect attacks is that most

CAN messages are periodic. This means that each ECU sends their required messages at regular intervals.

The IDS in [59] is an One-Class Support Vector Machine (OCSVM) which looks at several statistical features of the CAN messages. These features included the mean time difference between successive messages, the variance of the time difference and the number of packets sent amongst others. Using these statistical features, they were able to train their OCSVM to identify message injections and erasures. The paper [60] introduced their Generative Adversarial Network (GAN) based IDS (GIDS) to detect both known and unknown attacks. They turn the CAN message IDs into image data by one-hot encoding the hexadecimal values in the IDs. Since each ID consist of three hex values, the resulting one-hot encoding is of the shape 3 x 16. Afterwards, the IDs of several consecutive messages are concatenated to create a full input image. Their IDS consist of two separate "discriminators" which classify whether the input contains an attack or not. The first discriminator is trained in a standard way using a dataset containing known attacks and labels. The second discriminator is trained using inputs generated by a "generator" which is a GAN. A GAN is a deep learning network that generates fake inputs that resemble actual inputs. By training the second discriminator on inputs created by the generator, it learns to detect attacks it has not seen before (which is normally one of the weak points of supervised methods). The final IDS first feeds its input to the first discriminator, which determines whether it contains an attack or not. If it does not detect an attack with high confidence, the input is then forwarded to the second discriminator. This way known attacks are filtered out by the first discriminator, and potential unknown attacks can still be detected by the second discriminator.

Next, the paper [1] uses a similar method to transform message IDs into image data. Instead of one-hot encoding the hexadecimal values, this paper uses the binary representation of the message IDs and creates a "frame" of 29 consecutive messages. They choose 29 since their CAN messages use the extended format where the message ID consists of 29 bits instead of 11. These frames are used as an input for their IDS, which is a reduced version of the Inception-ResNet model. This is a state-of-the-art convolutional model used for image classification. They are able to detect various attacks such as DoS and spoofing using this method. Finally, the paper [61] proposes a graph based IDS. A window of 200 CAN messages is turned into a graph by turning the message IDs into nodes and connecting the IDs of consecutive messages with edges. Afterwards, several properties are extracted from the graph such as the adjacency lists and the degrees of the nodes. These properties are then compared to an attack free graph using the Chi-Squared test. This method allows the IDS to see if there are any anomalies in the number of messages of a specific ID as well as the order of messages in the 200 message window. Attacks such as DoS, Fuzzy and Replay can be detected using this method.

One downside of timing-based IDSs is their inability to successfully learn the behaviour of standard aperiodic CAN messages. Additionally, attacks that do not interfere with the normal timing of the CAN messages also tend to evade these types of IDSs.

Hybrid

Recently, research has started to investigate IDS models that are able to look at all CAN features. This allows them not only to detect a larger portion of the known attacks, but it also makes them more likely to detect unknown attacks. In [62] the IDS CANTransfer is introduced. This IDS uses convolutional LSTM layers, which combines the advantages of CNN and LSTM layers, allowing the model to effectively model multivariate time series data (data where multiple variables change over time). The main feature of CANTransfer is its potential regarding One-shot learning [62]. This entails that the model can learn to detect new attacks after being trained on only a single data sample of that attack. In the paper they show this ability by training the model only on DoS attacks and testing it on other attacks before and after One-shot learning. The model is able to increase its f1-score with 81% for the unknown attack after One-shot learning. While this still makes the model vulnerable to unknown attacks initially, it can be quickly adapted to detect this attack as soon as a single sample is available.

Other research has focused on including the interaction between messages with different IDs, such as [63] and [64]. CANet, proposed in [64], first uses a LSTM part for each message ID, similar to other methods discussed earlier. But instead of using the outputs directly, each LSTM output is saved in a ”joint latent vector”. Once a new message arrives only the part of the latent vector is updated that corresponds to the message ID. This latent vector is then fed to an AE part which aims to reconstruct the messages of all IDs. Then based on the reconstruction error, it is decided whether the message is an attack or not. By using information regarding all message IDs in the latent vector, dependencies between messages of different IDs can be captured. CANIntelliIDS from paper [63] also aims to link the messages with different IDs. This model starts with several convolutional layers followed by layers of Attention-based Gated Recurrent Unitss (AGRUs). The CNN part is able to retrieve various features from the inputs, afterwards the AGRU part can learn sequential and contextual information. This allows CANIntelliIDS to learn both information regarding the flow of data within a single message ID, as well as contextual information of this data when compared with other message IDs. Using these techniques, CANIntelliIDS is able to outperform several other models when detecting attacks such as DoS, Fuzzy and Impersonation.

2.4.2 Attacks on Intrusion Detection Systems

There are numerous attack types that adversaries can use in order to prevent IDSs from performing their tasks [26]. These types differ depending on the goal of the attacker. ”Overstimulation” attacks aim to create messages that deliberately cause the IDS to raise an alarm. This will cause whatever system or supervisor that is monitoring the IDS to be overwhelmed. Attackers can try to gain information regarding the IDS in order to apply ”Reverse Engineering” to recreate the employed IDS. This would allow them to create more advanced and specific attacks against this IDS, which would increase their success rate. Finally, attackers can aim to go unnoticed by the IDSs by either employing ”Evasion” or ”Poisoning” attacks. Both of these attack types aim to make the IDS misclassify their adversarial message as benign.

2.4.2.1 Poisoning

Poisoning attacks revolve around adding adversarial samples to the training data of the IDSs in order to lower their accuracy. These samples are created by using the original input data but matching them with incorrect labels. This way attack messages can be disguised as benign messages, which forces the IDS to learn these attacks as benign. Once the IDS is employed and actually encounters similar attack messages, it will misclassify them as benign. The main challenge of this type of attacks is that it is hard to gain access to the training data of a model while/before it starts training. That is why this attack type is mostly used against systems that perform online re-training. These systems perform re-training while employed to adapt to newly discovered attacks or changes in the general message behaviour. During these re-training instances, the IDSs are vulnerable to potential poisoning.

The paper [65] proposes a form of poisoning for VIDSs. As mentioned earlier, a VIDS creates a voltage profile/fingerprint for all the ECUs in order to detect whether a CAN message was sent by the correct ECU or not. Additionally, they periodically update these fingerprints since changes can occur due to various factors such as battery-life or temperature of the vehicle. During these updates, the IDS is vulnerable to potential poisoning of the voltage fingerprints. The attack method proposed in [65] involves two compromised ECUs working together to corrupt the voltage fingerprint of a victim ECU. During the re-training of the voltage fingerprints, the victim ECU and one of the attacker ECUs start sending a message with the victim's message ID at the same time. Since the messages are identical no error is detected on the CAN bus, however the voltage signal is now different from what it would be if only the victim was sending a message. At one point during transmitting the victim ID the attacker stops sending the message, since it cannot know the exact content of the victim's message to keep sending an identical message. At this point, the voltage fingerprint of the victim is successfully corrupted. Afterwards, when the attacker wishes to send a message using the victim's ID, it performs the exact same steps, however now the second attacker ECU is pretending to be the victim ECU and sends the injected message.

2.4.2.2 Evasion

While poisoning focuses on reducing the accuracy of the IDS before deployment (or during re-training), evasion attacks aim to reduce the accuracy after deployment. This is generally done by learning how to emulate normal behaviour better, or by leveraging the weakness of machine learning models to perturbations. An example of better emulating normal behaviour is shown in [66], where they propose an advanced masquerade attack that evades CIDSs. In a standard masquerade attack, the messages with a certain victim ID are dropped and an attacker injects messages with this ID instead. While these attacks do take the periodicity of these messages into account, they do not factor in the different clock-skews of the victim and attacker ECU. These standard masquerade attacks are therefore easily detectable by CIDSs. However, the authors of [66] noticed that the clock-skews that the CIDS use are purely computed based on the timing between messages. Since CAN is a broadcast channel, this computation can also be performed by the attacker and afterwards be manipulated. So the attacker can estimate the clock-skew of the victim with respect to their own clock, which can then be used to send their own messages to the IDS without being detected.

Research that shows the possibility of evading IDSs by exploiting the weakness to perturbations has been performed in standard networks [67] as well as in automotive networks [68]. In [67], the authors show that the accuracy of a Multi-Layer Perceptron (MLP) for standard network intrusion detection can be reduced by at least 20% by using the JSMA. The paper

[68] showcases their LSTM model for CAN intrusion detection, which is able to detect intrusion with 98% accuracy. Afterwards, they create adversarial samples using the FGSM and the Basic Iterative Method (BIM). These adversarial samples reach a success rate of 98% and 99% for each method, respectively. Following this, they are able to increase the robustness of their LSTM model towards these samples by performing re-training on these samples.

2.5 Summary

To summarize, modern vehicles include a large amount of electronics. However, due to the increase of connectivity and the simple network protocols, several security issues have emerged. To combat these issues, various solutions have been proposed from encryption methods to IDSs. These IDSs are an application of anomaly detection, where the goal is to detect abnormal behaviour on the Controller Area Network.

Several different types of IDS have been researched, focusing both on physical and digital features regarding CAN communication. Recently, the use of DL techniques for IDS have increased in popularity. But these underlying DL techniques are not invulnerable. They include several weaknesses, including a sensitivity towards perturbations and the transferability of these perturbations.

Research has shown various DL based IDSs for CAN. These IDSs use various techniques such as CNNs, LSTMs and AEs. Several papers have shown the presence of the previously mentioned weaknesses in DL based IDSs for networks such as standard IT and IoT networks. In contrast, the research regarding the weaknesses DL based IDSs for CAN is still limited. There are papers covering the weaknesses of the IDSs that focus on hardware features, such as the evasion of CIDSs. However the weaknesses of IDSs that focus on software features for CAN are not considered. This will be the main focus point of this thesis.

Chapter 3

Methods

This chapter will describe the research methods used during the thesis. First, the different steps in the research process will be covered. This is followed by a description of the source of the datasets, and a description of the characteristics of the datasets. The experimental design of the thesis will be presented, including hardware and software used, as well as what models and attacks were tested. Next, the reliability and validity of the presented methods will be evaluated. The methods and metrics used to quantify the results from the experiments will be shown. Finally, a description of how these results will be evaluated for future conclusions is presented.

3.1 Research Process

The research process can be divided into the following steps.

Step 1 Dataset collection and preprocessing

Step 2 Creation of baseline models

Step 3 Implementation of attack method and adversarial samples

Step 4 Re-creation of state-of-the-art models

Step 5 Execution of test runs

Step 6 Evaluation of results

Step 1 focuses on investigating various public datasets. Several factors were of importance during the final selection of the dataset. These factors included: whether the data came from real vehicles or simulations, what network attacks were included and the format of the datasets. After the final dataset was selected, several preprocessing steps were executed to make the data more suitable for DL. The details regarding the dataset and the preprocessing steps can be found in Section 3.2.

Step 2 revolves around creating two baseline models. These models have two functions. Firstly, their performance can provide a comparison to see whether the state-of-the-art models behave differently in the presence of adversarial samples than standard models. Secondly, the first baseline model, which will be described in Section 3.3, is used to generate the adversarial samples on.

In Step 3 the chosen attack method is implemented and adjusted to better adhere to the CAN specifications. This method is then used to generate the adversarial samples on the first baseline model.

Step 4 aims to re-create the state-of-the-art models. These models have been chosen to include different DL techniques. Additionally, both models focus on different data features, which may or may not help them in protecting against adversarial samples.

Next, in Step 5 the adversarial samples are tested on all the models. And the resulting performance of each model is collected. These results are then evaluated during Step 6 of the research process.

3.2 Data Collection

RISE guided me to various public datasets that were used in at least one paper. They suggested several different datasets because there currently is no benchmark dataset for CAN intrusions, such as there is for standard IT networks (think of KDD-99 dataset [69]). These datasets included the ORNL dataset [70], a dataset created by the Technical University of Eindhoven [71] and several datasets from the Hacking and Countermeasure Research Lab of South Korea [72, 60, 73]. The dataset that was eventually used was the Survival dataset presented by the Korean research lab in [73].

3.2.1 Survival dataset characteristics

The Survival dataset contains several logs of CAN traffic for three different vehicles: a HYUNDAI YF Sonata 2010, a KIA compact SUV Soul 2015, and a CHEVROLET mini-compact vehicle Spark 2015. For each of these vehicles, four different log files were recorded. One with only normal CAN traffic, one with DoS attacks, one with Fuzzy attacks and the final one with Malfunction attacks.

For the DoS attacks, several messages with the ID 0 were injected. This would result in other messages being pre-empted as per the arbitration rule described previously. For the Fuzzy attacks, messages with random IDs and random message data were injected. The goal of these attacks is to find out what combinations might have useful effects for the attacker. Finally, for the Malfunction attacks, specific message IDs were targeted, and by altering the message data unexpected behaviour resulted in the vehicles.

The format of the log files can be seen in Figure 3.1. The first field is a timestamp for when the message came in. Then the message ID is recorded in hexadecimal. This is followed by the DLC. Next, 0 to 8 data bytes are recorded, again in hexadecimal. Finally, a label of either R or T signalling normal and attack messages respectively. An example of the log file can be seen in Figure 3.2.

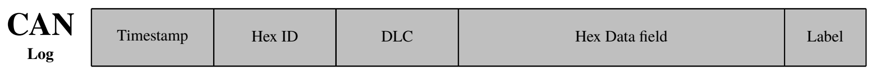
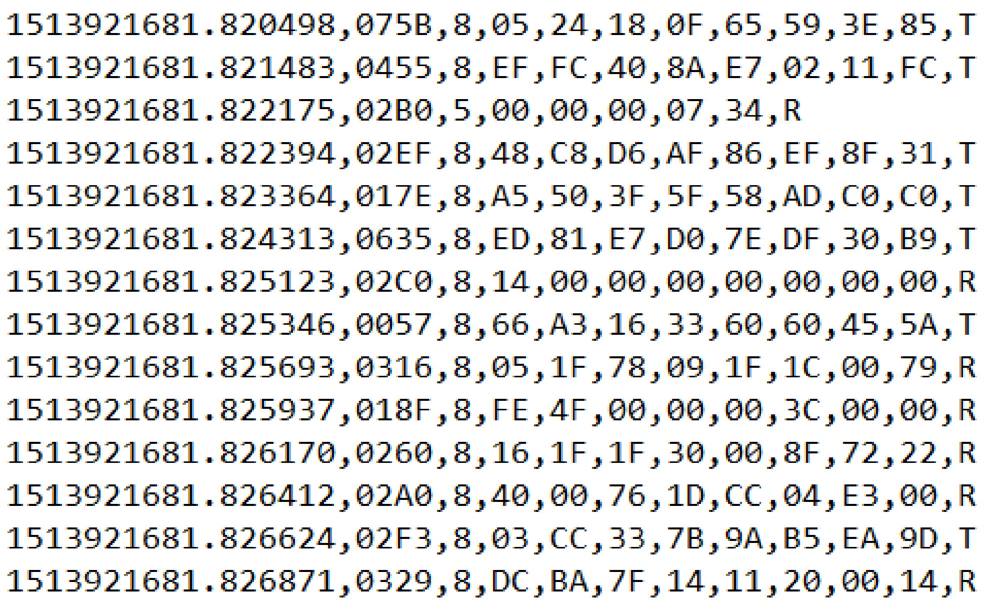


Figure 3.1: CAN log



```
1513921681.820498,075B,8,05,24,18,0F,65,59,3E,85,T
1513921681.821483,0455,8,EF,FC,40,8A,E7,02,11,FC,T
1513921681.822175,02B0,5,00,00,00,07,34,R
1513921681.822394,02EF,8,48,C8,D6,AF,86,EF,8F,31,T
1513921681.823364,017E,8,A5,50,3F,5F,58,AD,C0,C0,T
1513921681.824313,0635,8,ED,81,E7,D0,7E,DF,30,B9,T
1513921681.825123,02C0,8,14,00,00,00,00,00,00,00,R
1513921681.825346,0057,8,66,A3,16,33,60,60,45,5A,T
1513921681.825693,0316,8,05,1F,78,09,1F,1C,00,79,R
1513921681.825937,018F,8,FE,4F,00,00,00,3C,00,00,R
1513921681.826170,0260,8,16,1F,1F,30,00,8F,72,22,R
1513921681.826412,02A0,8,40,00,76,1D,CC,04,E3,00,R
1513921681.826624,02F3,8,03,CC,33,7B,9A,B5,EA,9D,T
1513921681.826871,0329,8,DC,BA,7F,14,11,20,00,14,R
```

Figure 3.2: Log of CAN messages with Fuzzy attacks

3.2.2 Preprocessing

The current format of the CAN messages is not directly suitable as inputs for DL techniques. To transform the data, several preprocessing steps were taken. First, all the hexadecimal values were translated to decimal values and later to binary values. The reason behind representing the values in binary, is that they will also be sent as binary values over the CAN bus. Additionally, this makes it easier to determine the size of the perturbations later during the thesis.

After all the values were translated to binary, a new data feature was added: delta Time (or dTIME). This feature was computed using the timestamps of the messages. The dTIME measured the time between the current message and the last message with the same ID. This feature was added since raw timestamps do not provide much information. However, the time between messages of the same ID provides much more information, especially since most CAN messages are periodic.

Next, the R and T labels were mapped to 0 and 1 values. Additionally, the labels for the normal data log were added, since they did not include any labels at all.

Finally, the various data logs were re-balanced. Since the number of attack messages was heavily outnumbered by the number of normal messages. If such a skewed dataset is used for the training of DL models, the resulting model will be biased towards the normal messages. Since it can get a high accuracy by simply predicting all the messages to be normal. For this re-balancing, random undersampling was used. This means that from the majority class (in this case the normal messages), only a subset is selected, such that the number of normal and attack messages is equal. Note that for the re-balancing, the SMOTE method was considered to up-sample the attack messages [74]. However, since SMOTE synthesizes new samples, this means that the values of the attack messages are different from the original attack message values. But since these values were specifically defined to create working attacks (except for the Fuzzy attacks), the SMOTE method was deemed unsuitable. Oversampling the attack messages by copying them was also ruled out since it could lead the models to overfit on the specific values of the attacks instead of the underlying characteristics.

3.3 Experimental design

3.3.1 Attacker Model

During the thesis, several assumptions regarding the attacker are made. The assumptions revolve around what actions are allowed and what information is available to the attacker during the creation of the adversarial samples. These assumptions represent the attacker model. This model helps substantiate the decisions made during the project.

The attacker model of this thesis is based on two different attacker modelling schemes for NIDSs [75, 76]. These schemes use different factors to describe the attacker's knowledge and capabilities.

3.3.1.1 Attacker's knowledge

The attacker's knowledge covers several aspects of the implemented IDS, such as what model type is used, the specific architecture and what data features it is trained on. Generally, the knowledge of an attacker is divided into three categories: white-box, gray-box and black-box. White-box means that all information regarding the model and architecture is known to the attacker. This would allow an attacker to make more sophisticated and targeted attacks. In black-box, none of the model specifications is known to the attacker. Gray-box lies in between black and white-box, meaning only a part of the characteristics of the model is known to the attacker.

In the thesis, it is assumed that the attacker has white-box knowledge of the first baseline model, while they have black-box knowledge of the other models (the specific models will be expanded on later). This assumption is made to simulate a ”oracle black-box” attack, such as described in [33]. In such an attack, the attacker has oracle access to the target system, meaning they can provide their own input to the target system and receive corresponding outputs. However any other information of the target system is unknown. By using the oracle functionality, the attacker can create their own substitute dataset which they can use to train a substitute model. Since all information of the substitute model is available to the attacker, they can use it to generate an attack, which can then potentially be transferred to the target system. In this situation, the attacker has white-box knowledge of their substitute model, but black-box knowledge of the target system. To match this oracle black-box attack, during the attack generation only the model parameters of the first baseline model can be accessed, while the parameters of the other models are off limits.

3.3.1.2 Attacker's capabilities

The attacker's capabilities are once again based on [76, 75] and it covers several aspects. Firstly, the type of access the attacker has to the system, in this case the vehicle. Secondly, the type of access the attacker has to the datasets used for training or testing and the IDS. Regarding the system, the assumption is that the attacker has compromised a single ECU and is now able to inject messages on the CAN bus. As mentioned previously, researchers have shown that an attacker only needs access to a single ECU to then gain control over the full vehicle [5]. For the datasets and the IDS, it is assumed that the attacker has no access to the training data, which will therefore not be altered during the perturbation generation. The reason that the training set is assumed to be inaccessible is to make the attack scenario of this thesis more realistic [76]. The training dataset is usually one of the company secrets and several measures are taken to keep it secret. This makes it difficult for an attacker to gain access to the training dataset. The assumption is made that the attacker will have access to a separate test set, since the deployed IDS can be used as an oracle to generate a test set as described earlier. Note that no restrictions on the oracle capabilities are assumed, such as a limited number of oracle requests. This is due to time constraints and to limit the scope of the thesis.

3.3.1.3 Attack scenario

This section will provide a potential attack scenario which shows how the experiments of this thesis can correspond to a real-life attack. The scenario starts with the attacker having compromised a single ECU in the target vehicle. The attacker uses the ECU to read CAN messages and inject messages such as DoS or Fuzzy attacks. This allows the attacker to use the IDS in the vehicle as an oracle, as describe earlier. After sending several messages and collecting the corresponding labels, the attacker can create a substitute dataset. Using this dataset, the attacker implements their own DL-based IDS. The attacker then uses their method to compute perturbations on their own IDS, and creates adversarial samples by adding these perturbations to the attack messages. Finally, he can use the compromised ECU to send the adversarial samples on the CAN bus.

3.3.2 Test environment

This section will provide a description of the chosen models and the attack method. The focus will lie on the theory behind these models and attack method as well as the role they play. Chapter 4 will provide the full description regarding the implementation details. Additionally, the different adversarial datasets and the reasoning behind them will be explained.

3.3.2.1 Machine Learning Models

As mentioned, four different model architectures were implemented during the thesis. The **first baseline model** is a standard DNN, from now on referred to as BL-DNN (short for BaseLine DNN model). The role of this model was to gain insights into the workings of IDSs and provide a baseline regarding the performance for future comparison between models. This model implemented several techniques that prevent overfitting on the training data, in order to make it as robust as possible. These techniques will be elaborated upon in Chapter 4. In addition, the adversarial datasets were all created on this model.

The **second baseline model** is an ensemble model consisting of 5 different ML techniques, from now on referred to as BL-Ensemble (short for BaseLine Ensemble model). These techniques are Logistic Regression, Decision Tree, Support Vector Machine, K-Nearest Neighbors and Naïve Bayes. These different techniques would each classify an input sample, thus voting for a certain class. The class with the most votes would then be returned as the output. The role of this model is to act as a second baseline regarding the performance. Additionally, it provides potential insights whether ensemble methods are more or less resilient towards adversarial samples.

The **first state-of-the-art model** is initially introduced in [1] and uses a deep convolutional network architecture. The model will be referred to as SOTA-CNN (short for State-Of-The-Art CNN model) to distinguish it from general CNN models. The architecture of this model is a reduced version of the Inception-ResNet model proposed in [77]. To use a CNN based model for the CAN messages, a certain transformation has to be implemented. The authors of this paper transformed several messages into frames. This was done by taking the message IDs of 29 consecutive messages and creating a matrix of these IDs. They used 29 messages, since their dataset used the extended CAN format, meaning that each message had a 29 bit ID. This resulted in square 29 x 29 frames.

These frames of message IDs allowed their model to detect various attacks based on their ID and timing. These attacks included DoS attacks, Fuzzy attacks and two types of ID spoofing. This paper focused primarily on the Error Rate and False Negative Rate for their metrics. The Error Rate is the opposite of the Accuracy, where the percentage of misclassifications is measured. While the False Negative Rate can be computed as $1 - Recall$. While the Recall metric looks at the percentage of positive predictions with respect to all positive samples, The False Negative Rate looks at the percentage of negative predictions with respect to all positive samples. They additionally showed the precision, recall and f1-score of their model on the different attacks. The model performed well in detecting the various attack types, especially on the DoS, and Spoofing attacks.

The Inception-ResNet model is based on two different CNN techniques, inception modules [78] and residual learning [79]. The idea behind inception modules is to replace fully connected convolutional layers with sparse layers. This would significantly reduce computational resources required. These sparse layers are actualized by using different smaller convolutional layers instead of single larger layers. The second technique of residual learning works as follows. Generally, a neural network aims to approximate a mapping from the inputs to the desired outputs. However, in residual learning, the mapping that is learned is the residual that remains from subtracting the inputs from the original mapping. This is done by introducing shortcut connections, where (intermediate) inputs are fed forwards without going through any neurons to be added to future intermediate values. These residual mappings are easier to optimize and allow for better performance on deeper networks than non-residual mappings.

The **second state-of-the-art model** is based on an LSTM and was proposed in [56]. This model will be referred to as SOTA-LSTM (short for State-Of-The-Art LSTM model) from now on. The SOTA-LSTM architecture is divided into two parts. The first part includes the LSTM layers and aims to predict the next input in a sequence given the current input. This part is trained in a unsupervised manner, by only providing it with normal message data and letting it predict the desired next message. The goal is to minimize the difference between the predicted next input and the actual next input. The second part of the model performs the actual classification. An input is given to the first part of the model, which then returns the predicted next input. The error between the predicted and actual next input is then compared to a certain threshold. If the error is above the threshold, the input is classified as an attack, otherwise the input is classified as a normal message. This threshold is set to the 99th percentile of the errors that originated from the training phase.

In contrast to the previous model, this model does not look at the message IDs at all, but instead only focuses on the data bytes of the messages. The IDS aims to model the data flow of the messages, and thus find any messages that do not adhere to this flow. Since the messages of each ID have their own data flow, the paper constructed several SOTA-LSTM models, one per message ID. This allows each model to learn the data flow of a single message ID.

The attacks that this model is designed to detect all disrupt the data flow in some manner. First the ”interleave” attack, takes two valid sequences of message data and interleaves them. Thus creating an abnormal data flow. The ”drop” attack, removes a set amount of messages from a sequence. In the ”discontinuity” attack, a new sequence is created by taking the first half of one sequence and the second half of another sequence from a later time. Both sequences are valid, however the switch that occurs from one to the other is not valid. The ”unusual” attack alters the bits from various messages in the sequence. Based on data analysis, the researchers discovered that certain bits were generally unused by certain message IDs, so the focused on including these bits in the ”unusual” attack. Finally, the ”reverse” attack takes a valid sequence, but reverses it.

These different models were chosen/designed such that they use different techniques. This will provide the most information regarding how the response to adverserial samples differs per technique. Additionally, the state-of-the-art models both focus on different features of the CAN messages. This in turn can show whether certain features might be more robust than others.

3.3.2.2 Attack method

The method that is used to generate the adversarial data is called the Fast Gradient-Sign Method (FGSM) [28]. The work flow of perturbation generations is easier to understand when it is compared to the work flow of standard gradient descent used in machine learning. As can be seen on the left in Figure 3.3, gradient descent works as follows. First the input is fed to the model, after which the output and the corresponding loss is computed. Then during gradient descent, the gradient of the loss is taken with respect to the model weights. This gradient is then used to update the weights of the model to minimize the loss. For the FGSM, the first steps are identical. However, now the gradient of the loss is taken with respect to the input. This gradient is then used to compute the perturbation and update the inputs instead of the model weights.

The FGSM computes the perturbations as

$$\eta = \epsilon * sign(\nabla_x J(\theta, x, y)), \tag{3.1}$$

where the $J(\theta, x, y)$ is the loss function of the model on which the perturbations are generated, θ are the model parameters, x is the inputs and y

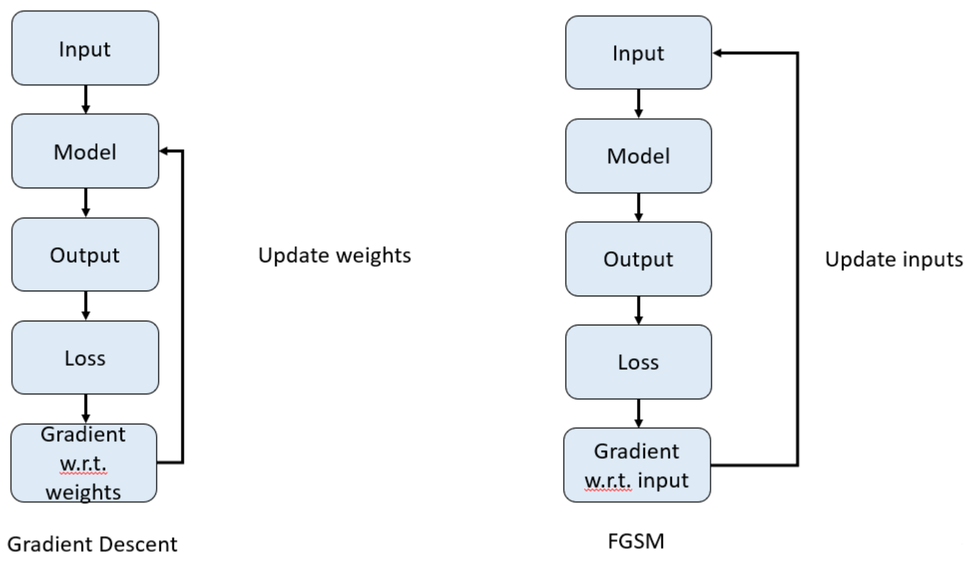


Figure 3.3: Comparison of the work flow of FGSM and gradient descent

are the corresponding outputs. The gradient of this loss function is computed with respect to the inputs. After the gradient is computed, the sign of this gradient is taken, which is represented by an array of -1s, 0s and 1s depending on the sign of the values in the gradient. This sign is then multiplied with ϵ, where the ϵ defines the size of the perturbation. The sign of the gradient is directly multiplied with ϵ since the original FGSM sets the magnitude of the perturbation based on the L-inf norm.

Note that for the thesis some alterations have been made to the method which will be covered in Section 4.1, but the underlying principles have not changed. Additionally, the specific datasets that were created using this method will be discussed in Chapter 4.

3.3.3 Hardware/Software to be used

The training and testing of the models was done via RISE's datacenter called ICE (`https://ice.ri.se/`). The datacenter provided access to a NVIDIA-GTX-2080ti GPU.

Regarding software, the project was executed using the Python programming language in Jupyter Notebooks. The main packeges that were used were: Numpy, Pandas, Imblearn, Keras, Scikit-learn, Tensorflow and Cleverhans. Additionally, the Matplotlib.pyplot and the Seaborn packages were used for the plots and visualizations.

3.4 Validity of the methods

The validity of the methods comes from the systematical research steps and the pre-established validity of the chosen models and attack method. By adhering to the general steps of designing, training and testing DL models, the validity of these final models can be guaranteed. The grid search method was used to find the set of optimal parameters for each model, while the architectures of the models are identical to the previous models in [1, 56]. Hence, the validity of the original architecture is maintained, while the best performance of IDSs given the architecture is obtained. Regarding the creation of the adversarial samples, only additional restrictions have been applied to the original FGSM, thus retaining the underlying functionality and the accompanying validity.

The models will be executed on the standard datasets and those with adversarial samples. The execution method will be equal for both types of datasets, assuring that the results can be compared accurately. The final comparison will be used to support or oppose the hypothesis of this thesis.

3.5 Planned Data Analysis

The planned data analysis of the thesis can be divided into two sections. The first section is the analysis of the performance of the models on the adversarial samples. The second section is the analysis of the perturbations of the different adversarial datasets.

3.5.1 Performance Analysis Technique

For the performance analysis, several metrics that revolve around the number of correctly and incorrectly classified samples will be used. The main metrics/techniques are a confusion matrix, accuracy, precision, recall and f1-score.

First, a confusion matrix can be seen in Table 3.1. In this case, the negative samples refer to normal messages, while the positive samples refer to attack messages. The confusion matrix divides the samples into four groups being: True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN). True positive counts the number of samples that have the positive class and are predicted to be positive by the model. False positive counts the samples that are negative, but are falsely predicted to be positive. True negative counts the negative samples that are predicted to be negative. False Negative counts the positive samples that are predicted to be negative. This gives a quick overview of how many samples are classified correctly and what types of mistakes are made by the model.

		Predicted class	
		Negative	Positive
True class	Negative	TN	FP
	Positive	FN	TP

Table 3.1: Example confusion matrix

Using these four groups of data points, several other metrics can be computed. The equations for the accuracy, precision, recall and f1-score are

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}, \tag{3.2}$$

$$precision = \frac{TP}{TP + FP}, \tag{3.3}$$

$$recall = \frac{TP}{TP + FN}, \tag{3.4}$$

$$f1 - score = 2 * \frac{precision * recall}{precision + recall}. \tag{3.5}$$

The accuracy provides a single number, signifying what percentage of all the samples are classified correctly. This metric provides a quick look on the performance of a model, however it can provide a biased result. If the original dataset is unbalanced, for example 80 normal messages and 20 attack messages, the accuracy of a model can be 0.8 if it predicts all messages to be normal. However, this model is unusable in practice, since all attack messages are missed.

To cover the weakness of the accuracy metric, the precision and recall metrics are used. The precision metric computes what percentage of the samples predicted to be positive is actually positive. While the recall metric computes what percentage of the positive samples is predicted to be positive. Precision focuses on the correctness of the predictions, and recall focuses on the completeness of the predictions. Additionally, the f1-score computes the harmonic mean of the precision and recall. This transforms these two metrics into a single value, which makes it easier to evaluate the model.

3.5.2 Perturbation Analysis Technique

To analyze the perturbations, several factors will be taken into account. The number of iterations needed to create each adversarial sample will be investigated. The number of iterations represent the number of times the FGSM is called until a successful adversarial sample is generated or until a iteration limit is reached. The total size of the perturbations will be evaluated. Finally, the specific perturbation values will be investigated.

The number of iterations and the size of the perturbations provide insights into how close the original samples lie to the decision boundary. And by extension, how robust the models are with respect to those specific samples. For the specific perturbations, the attention will be on what bits or other values are most often changed. This shows which bits are more influential during the classification.

3.6 Evaluation framework

To evaluate the final results, the metrics described previously will be computed for each model on both the adversarial datasets and the standard datasets. The values of these metrics will show whether the perturbations are capable of influencing the performance of the models. Additionally, since all adversarial samples are only generated on the first baseline model, the transferability will be shown by observing the influence of the samples on the other models.

Additionally, the performance and the drop in performance of the different models will be compared to see what models are more resilient.

Chapter 4

Code and Implementation details

This chapter provides a deeper look into the code and implementation details relevant to the thesis. First, the FGSM is shown in psuedo-code format, and any alterations made will be described. The different datasets that were generated with the FGSM are described, as well as the reasoning behind the chosen features for each dataset. Finally, the architectures and parameters of each of the models used during the thesis is covered.

4.1 Fast Gradient-Sign Method

For the FGSM, the basic implementation has been copied from the Cleverhans library. The psuedo-code can be seen in Algorithm 1. The Cleverhans functionality is displayed with the black text, and the extensions to the method are shown in red text. The corresponding Python code can be seen in Appendix A.

The input parameters for the function are as follows:

- **model_fn:** The model on which to generate the perturbation (note: this model should return logits instead of the final class probabilities to effectively compute the gradient).
- **x:** The input that is to be perturbed.
- **eps:** The size of the perturbation measured using the norm.
- **norm:** The norm to be used to compute the size of the perturbation.

Algorithm 1 Fast Gradient Sign Method

```
Require: model_fn, x, eps, norm, features, loss_fn, clip_min, clip_max, y
# Compute the gradient of the loss_fn on model_fn w.r.t. x
grad = compute_gradient(model_fn, loss_fn, x, y)

# Limit gradient to chosen features
limit_grad = features * grad

# Further limitations
limitations = array of 1s of shape (len(features))
for i=0 to len(limit_grad) do
    if feature has max value and will be increased further then
        limitations[i] = 0
    else if feature has min value and will decrease further then
        limitations[i] = 0

limit_grad = limit_grad * limitations

# Compute perturbation based on gradient, epsilon and the desired norm
optimal_perturbation = optimize_linear(limit_grad, eps, norm)

# Add perturbation to original sample
adv_x = x + optimal_perturbation

# Clip values of adversarial sample to max and min values
adv_x = clip_by_value(adv_x, clip_min, clip_max)
return adv_x
```

- **features:** An array of 0s and 1s to select which features to include in the perturbation.
- **loss_fn:** The loss function to be used, defaults to sparse softmax cross entropy.
- **clip_min:** A single value or an array containing the minimum allowed values of the adversarial sample.
- **clip_max:** A single value or an array containing the maximum allowed values of the adversarial sample.
- **y:** The label to be used for the loss computation. This variable can be set to desired class for the adversarial sample, or by default to the actual predicted class in which case the loss is maximized instead of minimized.

The first extension can be seen in lines 4 and 5. It enables the option to select which input features to include during the perturbation generation. By multiplying the gradient with an array of 0s and 1s, unwanted features can be removed from the generation process. Lines 7 to 15 show the second addition, which is an additional check to see if one of the input features has already reached the minimum or maximum value, and will be updated again in that direction according to the gradient. Since these updates will be reversed once the final step of clipping the values is reached, a certain loop might appear. This is especially noticeable when using the L1-norm during perturbation generation. For example, let us say that feature 5 has the highest positive gradient value. This means that when using the L1-norm, this feature will be increased during the perturbation generation. However, if feature 5 is a bit value of 1, a higher value is not valid and will be clipped. Meaning that the update is reversed in the final step. Now the resulting output is equivalent to the original input, meaning no perturbation has been returned.

During the generation process, depending on what norm was used and what value for epsilon was taken, the success rate of the perturbation generation varied significantly. Whilst taking inspiration from the Basic Iterative Method [80], an additional loop function was implemented that would run the FGSM until a successful perturbation was generated or until a set amount of iterations was reached. This method can be seen in Algorithm 2.

The input parameters for FGSM_Loop are the same as for FGSM, with the addition of the max_iter parameter, which dictates the maximum number of iterations allowed for the perturbation generation. First, several necessary

Algorithm 2 FGSM Loop

```
Require: model_fn, x, eps, norm, features, loss_fn, clip_min, clip_max, y, max_iter
 1: # Setup necessary variables
 2: iteration = 0
 3: adv_x = x
 4: x_round = adv_x
 5:
 6: while x_round is not a successful adversarial sample and iteration < max_iter do
 7:     #compute adversarial sample
 8:     adv_x = FGSM(model_fn, adv_x, eps, norm, features, loss_fn, clip_min, clip_max, y)
 9:
10:     # if sample misclassifies, round values and check again
11:     if label of adv_x == y then
12:         x_round = adv_x rounded except dTIME value
13:
14:     # Update iteration counter
15:     iteration = iteration + 1;
16:
17: # Add check if maximum iterations reached
18: if iteration == max_iter then
19:     # update to latest sample
20:     x_round = adv_x rounded except dTIME value)
21: return x_round, iteration
```

variables are initialized. Next, in a loop, an adversarial sample is generated. This sample uses a perturbation specified by the norm and epsilon of the input parameters. If the perturbation causes the desired misclassification, the features of the sample that expect discrete values are rounded. This is necessary since the majority of the input features require discrete values (for example all the data bits), but depending on the choice of norm and epsilon the perturbations could include continuous values. After rounding the values, the new classification is computed since it might have changed. Next, the iteration counter is updated and if either the maximum number of iterations is reached, or the rounded sample misclassifies as desired, the loop is ended. The final rounded sample and the number of iterations it took to create the sample is returned. Note that if the sample is not generated within the specified number of iterations, the latest sample is rounded and returned.

4.2 Datasets

During the project several different datasets were created both for training and testing purposes, as well as for adversarial data generation. Each of the adversarial datasets used different parameters and features when generating the perturbations. Additionally, both the SOTA-CNN and SOTA-LSTM model needed their datasets to be in a special format. So the alterations made to these datasets will be mentioned in this section as well.

4.2.1 Training and Testing

The original datasets consisted of four different data logs, one for normal messages, one containing normal and DoS messages, one with normal and Fuzzy messages and one with normal and Malfunction messages. To create the training and testing datasets, all logs were used except for the pure normal data log. This decision was made since the datasets were already unbalanced, so adding more normal messages would be counter-productive.

For the other three logs, the data was preprocessed using the methods mentioned in Chapter 3. Thus a dTIME field was created, the ID and the data bytes were transformed into bit values and the logs were balanced using random undersampling. Afterwards, each of the 3 logs was split in a 60-40 ratio for the training and test sets respectively. The final training and test datasets were created by concatenating the three training and test splits. This resulted in the final training set having around 80,000 datapoints and the test set having around 53,000 datapoints. The distribution between normal and attack messages in both datasets is approximately equal (there are slight differences due to the way the split was created for each dataset).

4.2.2 Adversarial datasets

For each of the adversarial datasets that were generated, several parameter values used in the FGSM were decided on. The majority of these parameter values were identical for all datasets. All adversarial samples were generated on the BL-DNN model. The norm and epsilon values used are L1 and 1.0 respectively. This ensures that only 1 of the features is altered in every iteration with a step size of 1.0. The minimum value of all input features is set to 0, to prevent negative values. And the maximum value of all bit features (i.e. the ID bits and the 64 Data bits) is set to 1. The maximum value for the DLC and dTIME features are set to 8 and 100 respectively. Finally, the maximum number of iterations is set to 50, and the target for the adversarial samples is set to 0 (such that the attacks are classified as normal messages).

The only parameter values that differ per adversarial dataset are which features are included during the perturbation generation. The possible features are the 11 ID bits, the DLC, the dTIME and the 64 Data bits. Which features are used for each dataset will be described below and shown schematically for easier comprehension. The schematic image will show for each feature whether it is included or excluded as shown in Figure 4.1.

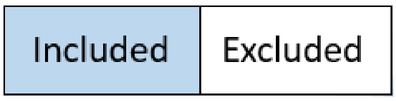


Figure 4.1: Legend for included and excluded features

4.2.2.1 Full adversarial

The Full adversarial dataset was generated based on the previously described test set. For this dataset, all features were included in the generation of the perturbation. The features can also be seen in Figure 4.2. A figure showing the complete parameter list can be seen in Appendix B.

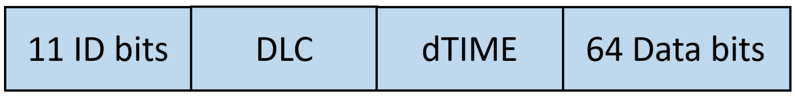


Figure 4.2: Included features for FGSM for Full Dataset

4.2.2.2 DoS adversarial

For the DoS adversarial dataset, only the test split of the DoS log was used. The main aim of these adversarial samples is to evade detection whilst still being able to function as the original attacks. DoS attacks rely heavily on two factors: a low message ID (e.g., a decimal ID of below 10) and a high transmission rate (i.e., a low dTIME). Therefore, when generating the perturbations for the DoS attacks, the dTIME is left out as a feature entirely, and the message ID is capped at a maximum value of 7 by excluding all ID bits except for the final three. This way, only the final 3 ID bits have a chance of being perturbed from 0 to 1, meaning that the maximum value for the ID is 7 when all 3 ID bits are perturbed. The included features can be seen in Figure 4.3.

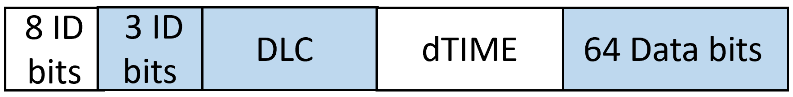


Figure 4.3: Included features for FGSM for DoS Dataset

4.2.2.3 Fuzzy adversarial

Similar to the DoS adversarial dataset, only the test split of the Fuzzy log was used for the Fuzzy adversarial dataset. In Fuzzy attacks, the main goal is to find combinations of message ID and data bits that result in unexpected behaviour. Therefore, the perturbations will focus on these features and the DLC and dTIME will be left out. The included features can be seen in Figure 4.4.

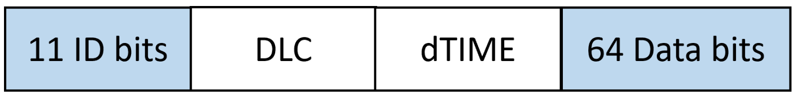


Figure 4.4: Included features for FGSM for Fuzzy Dataset

4.2.2.4 Malfunction adversarial

For the Malfuction adversarial dataset, the test split of the Malfunction log was used for perturbation generation. One key aspect of the malfunction attacks was that they targeted a specific message ID. Additionally, the messages were sent at specific intervals and always used 8 data bytes. Therefore, the message ID bits, the DLC and the dTIME were all excluded during perturbation generation. The included features can also be seen in Figure 4.5

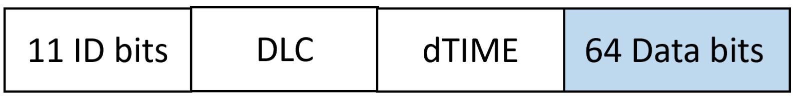


Figure 4.5: Included features for FGSM for Malfunction Dataset

4.2.3 SOTA-CNN datasets

The datasets that will be used will be different for the SOTA-CNN model in comparison with the other models. This is since the model expects the data to be in the form of frames. The psuedo-code for the frame generation function can be seen in Algorithm 3 and Algorithm 4. Note that this is originally one function, but it is split up to increase readability. The corresponding Python code can be seen in Appendix A.

Algorithm 3 Frame Builder for the SOTA-CNN model data (Part 1)

```
Require: $X, Y, nrAttacks$
length = nr of frames that can be created from the dataset

# setup required variables
X_frames = array of 0s of shape (length, 29, 29)
Y_frames = array of 0s of shape (length)

# Setup Initial Frame
initial_frame = array of 0s of shape (29, 29)
attack_counter = 0.0
initial_label = 0.0
for i=0 to 28 do:
    initial_frame[i] = bit ID of X[i]

    #add label values
    attack_counter = attack_counter + Y[i]

if attack_counter > nrAttacks then
    initial_label = 1.0

Save initial_frame in X_frames
Save initial_label in Y_frames
```

The frame builder creates frames by combining 29 consecutive message IDs. Note that the original dataset used in this thesis has message IDs of 11 bits, so the other subsequent bits are padded with 0s. This does not remove any information from the message IDs and it also does not add any other information to the frames. The label of each frame is determined by the amount of attack messages in the frame.

Since the previously described training and test set are balanced, the only way to gain a balanced frame dataset is by setting the attack threshold to 14

Algorithm 4 Frame Builder for the SOTA-CNN model data (Part 2)

```
Require: $X, Y, nrAttacks$
22: # compute other frames based on initial frame
23: temp_frame = copy(initial_frame)
24: for each remaining row in X do
25:     temp_label = 0.0
26:     # remove first row and adjust attack counter accordingly
27:     attack_counter = attack_counter - (label of first row)
28:     temp_frame = temp_frame with first row deleted
29:
30:     # add new row and adjust attack counter
31:     attack_counter = attack_counter + (label of new row)
32:     temp_frame = temp_frame with new row appended
33:
34:     if attack_counter $>$ nrAttacks then
35:         temp_label = 1.0
36:
37:     Save temp_frame in X_frames
38:     Save temp_label in Y_frames
39: return X_frames, Y_frames
```

attacks. However, if the IDS only classifies frames as attacks if more than half of the messages in the frame are attacks, various attack messages will be missed in the future. Since certain attacks can be quite sparse. Preferably, if 5 messages in the frame are attacks, it should already be considered an attack. But setting the threshold to 5 with the previously balanced datasets results in a heavily biased frame dataset where most frames are considered attacks.

To solve this problem, the previously unused train and test split of the Normal log is concatenated to the training and test dataset. These new datasets are then fed to the frame builder, which results in a fairly balanced training and test frame dataset. Where the training set has around 70,000 normal frames and 80,000 attack frames. And the test set has 46,000 normal frames and 53,000 attack frames. Since the Full adversarial dataset is equal in distribution to the test set, the same steps were executed to balance the Full frames dataset.

To create a similar balance for the DoS, Fuzzy and Malfunction dataset, respectively 25,000, 15,000 and 12,000 samples of the test split of the Normal log were added to the datasets. Once again creating frames with the attack threshold set to 5 gives the following distributions. The DoS frames dataset has around 25,000 normal frames and 26,000 attack frames. The Fuzzy frames dataset has 15,000 normal frames, and 14,000 attack frames. And the Malfunction frames dataset has 12,000 normal frames and 13,000 attack frames.

4.2.4 SOTA-LSTM datasets

The datasets also have to be adapted for the SOTA-LSTM models. Firstly, since one SOTA-LSTM model is trained per message ID, the datasets have to be filtered based on those IDs. One issue was that there were only a limited amount of message IDs that had a decent number of messages related to them. In the Normal log, there were only 17 message IDs that had more than 5000 messages each. This will come into play later since the adversarial datasets have to be re-balanced again.

From these 17 message IDs, the following 5 were randomly selected: 2, 305, 704, 809, 1088. For the training and testing datasets, all logs were used and split 60-40. For each model, only the messages of the corresponding ID were selected. Additionally, the SOTA-LSTM models only use the data bits in the messages, so the other features were dropped.

As mentioned earlier, the SOTA-LSTM models consist of two parts. The first part takes the data bits from the current input, and predicts the data bits of the next input. So for the training of the models, the training and testing set were transformed such that inputs and the labels consisted of the same databits. However the labels were shifted one spot such that the first message in the labels is the second message in the outputs.

After the models were trained, the loss on the training set was computed, and the 99th percentile of the loss was set as the classification threshold.

For the adversarial datasets, in addition to the messages of the chosen IDs all the attack messages were also included. Since the attack messages outnumber the normal messages, the attack messages are randomly undersampled. So for each adversarial dataset, first an input set was created. Then a predicted output set was generated, which was identical to the input set except that it was shifted by one place. Finally the labels of each message was collected for the final classification.

4.3 Models

In this section, the different models and design decision will be discussed. This includes the general architecture as well as what hyperparameters were used.

4.3.1 BL-DNN

The first baseline model is a standard DNN, where the main focus was put on preventing overfitting on the training data. The model consists of three dense layers with 128, 64 and 2 neurons respectively. These layers are separated by a normalization layer, which normalizes the intermediate outputs. Additionally, the first two dense layer uses the ReLu activation function and applies the L2 weight regularizar. The regularizer prevents the weights of the neurons from getting too large of a magnitude. The final dense layer uses the softmax activation function to perform the final classification.

All the weights of the model are initialized using TensorFlow’s default initializer. This is the uniform glorot initializer, where the weights are drawn from a uniform distribution whose limits are dependent on the number of input and output connections. The model uses the Adam optimizer with a learning_rate of 0.001 and has the Sparse Categorical Crossentropy as the loss function.

Finally, the model is trained for 10 epochs. However, early stopping is enabled which stops the training if the validation accuracy does not improve for 2 epochs. A schematic image of the structure of the model can be seen in Figure 4.6.

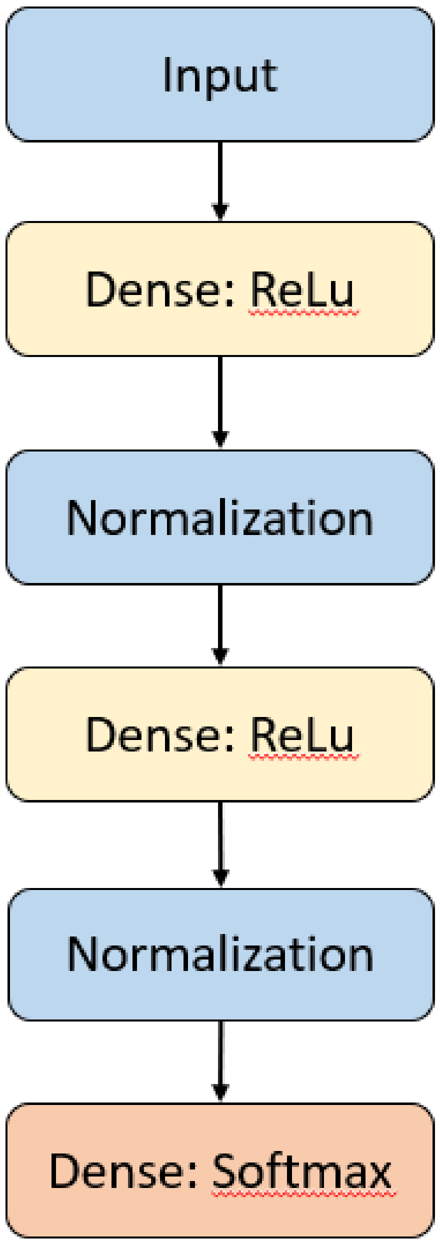


Figure 4.6: BL-DNN schematic

4.3.2 BL-Ensemble

As described in Chapter 3, the ensemble method consists of 5 different ML techniques. The techniques and the parameter settings used, are as follows.

- **Logistic Regression:** This model has a maximum amount of iterations of 1000 and uses the L2 norm to compute a penalty on the model parameters. The other parameters are the default of the Sklearn package, such as the lbfgs solver.
- **Decision Tree Classifier:** The maximum depth is set to 3. The other parameters are left as default as defined in the Sklearn package. Meaning the criterion used to determine the decision nodes is the Gini impurity.
- **Support Vector Classifier:** The model uses a linear kernel. All the other parameters are set to the default values defined by the Sklearn package.
- **K-Nearest Neighbor Classifier:** The number of neighbors is set to 5. The distance metric is set to Minkowski with a p value of 2, which is equivalent to the Euclidean distance metric.
- **Gaussian Naïve Bayes:** All the parameters are left on their default settings defined by the Sklearn package.

These 5 submodels are combined into a Voting Classifier, which uses hard voting. This means that the label predictions of each submodel are collected, and the majority vote becomes the final predicted label. A schematic overview of the BL-Ensemble model can be seen in Figure 4.7.

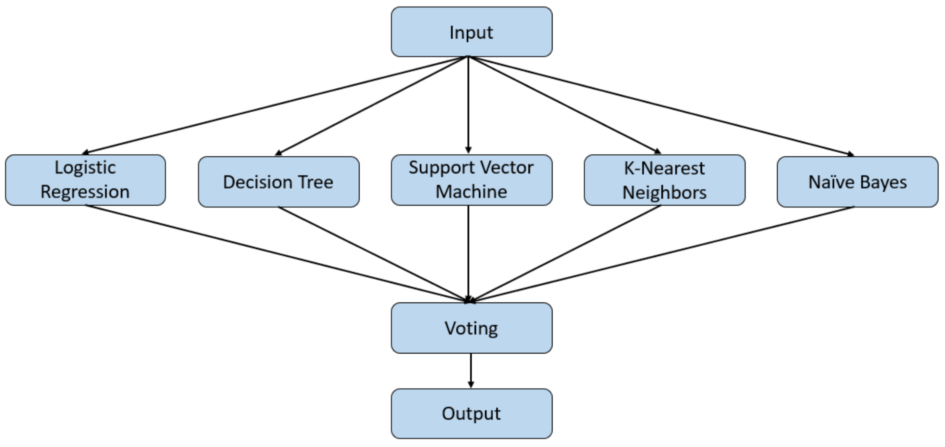


Figure 4.7: BL-Ensemble schematic

4.3.3 SOTA-CNN

The SOTA-CNN model is the most intricate of the implemented models. It consists of the following parts: a Stem block, a Inception-ResNet-A block, a Reduction-A block, Inception-ResNet-B block, a Reduction-B block, an Average Pool layer, a Dropout layer and a final dense softmax layer. A schematic of this structure can be seen in Figure 4.8.

The Stem block can be seen in Figure 4.9. It consists of several convolutional layers and a max pooling layer. Note that each convolutional layer in the whole model is always followed by a batch normalization layer.

After the Stem block, the inputs enter the Inception-ResNet-A block, which can be seen in Figure 4.10. This figure shows the several different branches of the Inception-ResNet block. The different branches are fed to later layers, where they are concatenated, fed to a convolutional layer and finally combined with a shortcut connection as is the standard with residual networks.

Next, the intermediate results go the the Reduction-A block, which can be seen in Figure 4.11. This layer reduces the dimensionality of the results via various convolutional and pooling layers.

Afterwards, the results are forwarded to the Inception-ResNet-B block, shown in Figure 4.12. Similar to the Inception-ResNet-A block, it consists of several branches which are then concatenated. Afterwards, the Reduction-B block follows in Figure 4.13.

Finally, the intermediate results are fed to the last pooling, dropout and dense layer. The model uses the Adam optimizer with a learning_rate of 0.001 and the sparse categorical crossentropy as the loss function. It is trained for 10 epochs, with the option of early stopping if the validation accuracy does not improve for 3 epochs.

To show the difference in scale with the BL-DNN model, while the BL-DNN model has around 18.000 trainable parameters, the SOTA-CNN model has around 1.7 million trainable parameters. Note that all images of the SOTA-CNN architecture are licensed and sourced from [1]. Several images of the implementation of this architecture can be found in Appendix C.

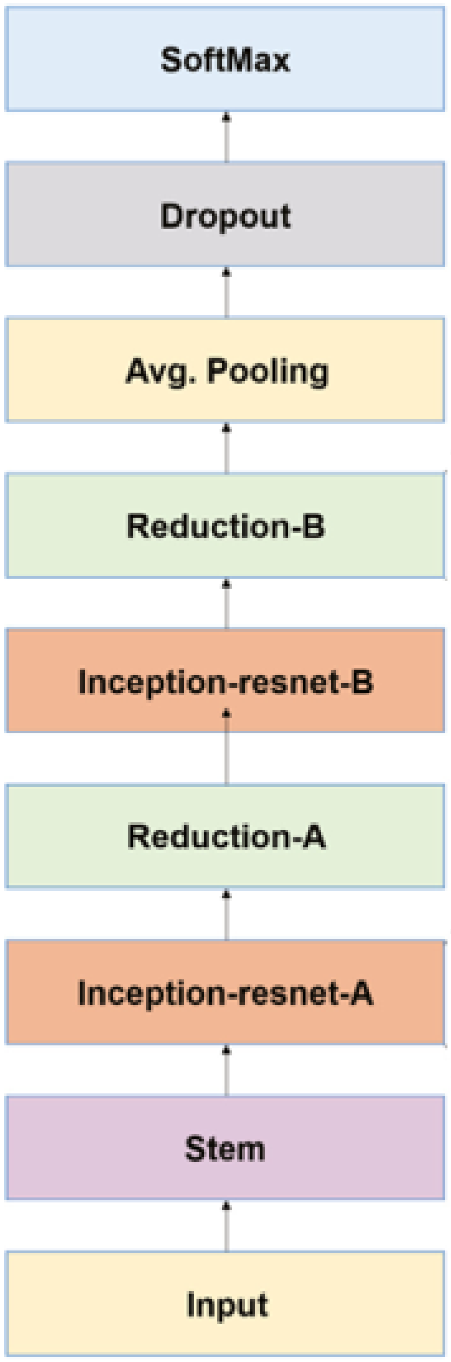


Figure 4.8: SOTA-CNN schematic

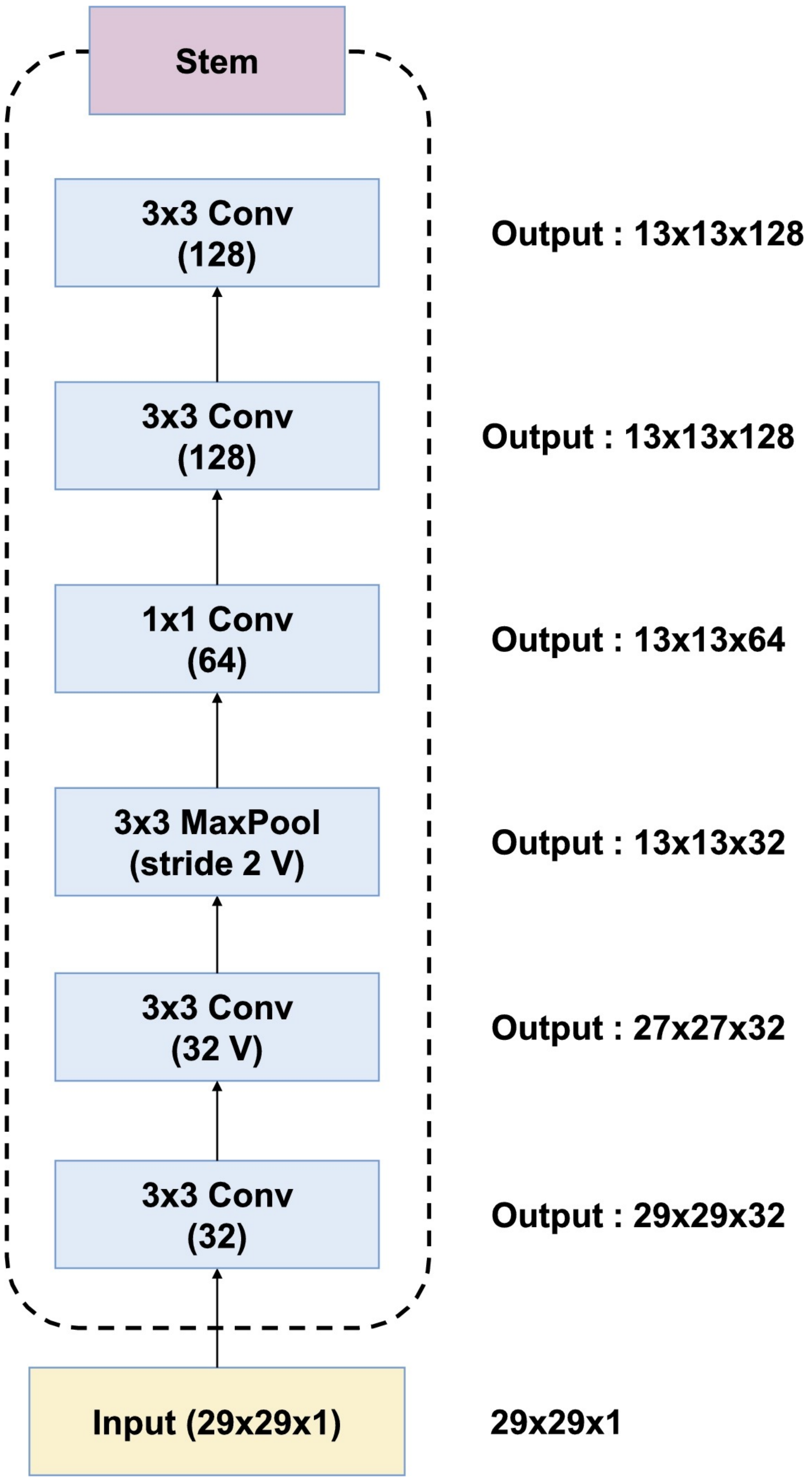


Figure 4.9: SOTA-CNN Stem block

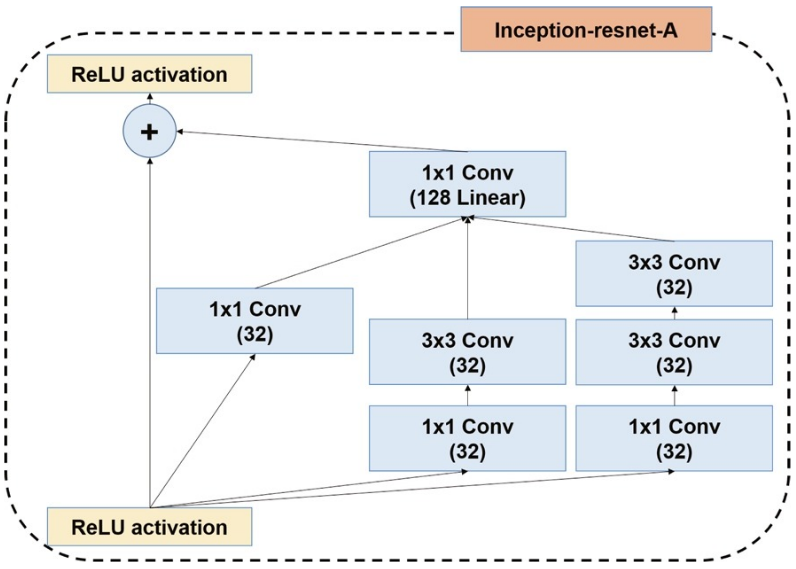


Figure 4.10: SOTA-CNN Inception-ResNet-A block

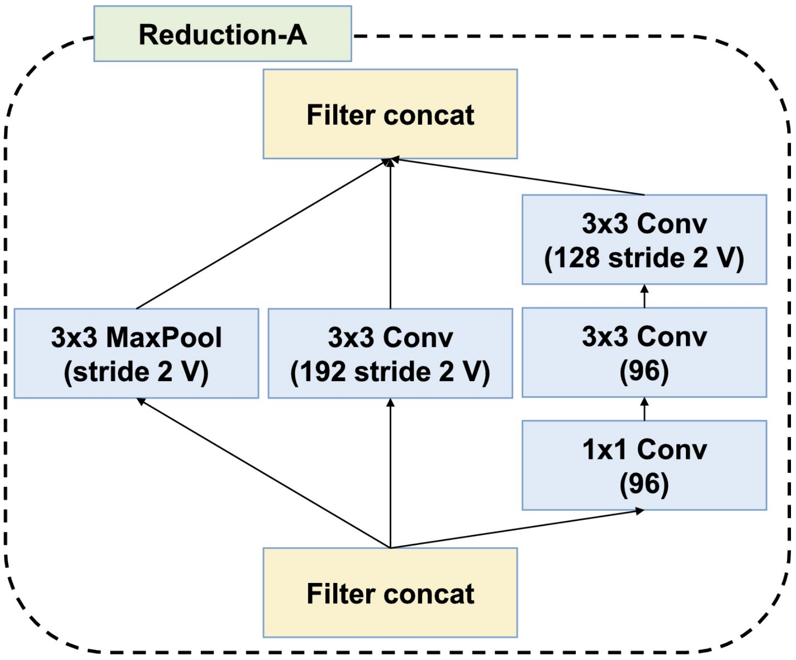


Figure 4.11: SOTA-CNN Reduction-A block

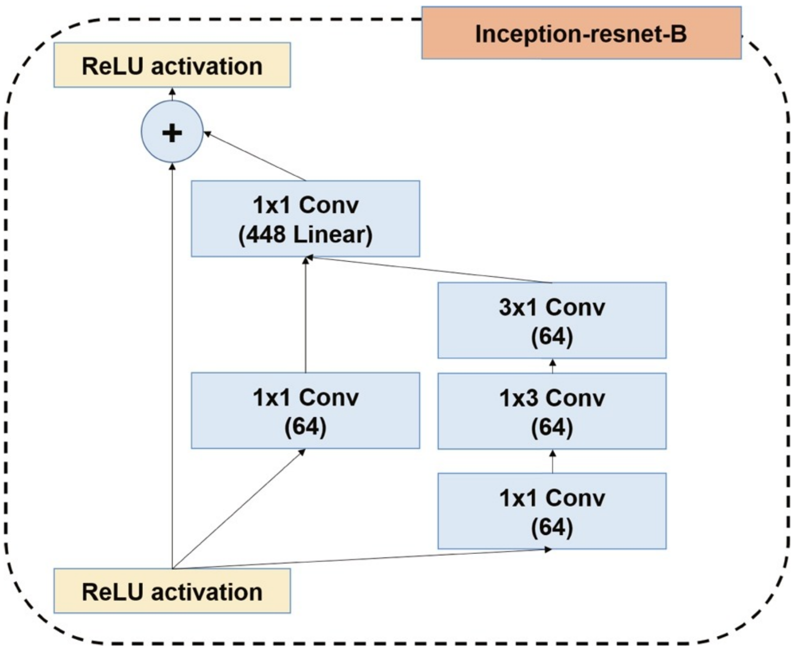


Figure 4.12: SOTA-CNN Inception-ResNet-B block

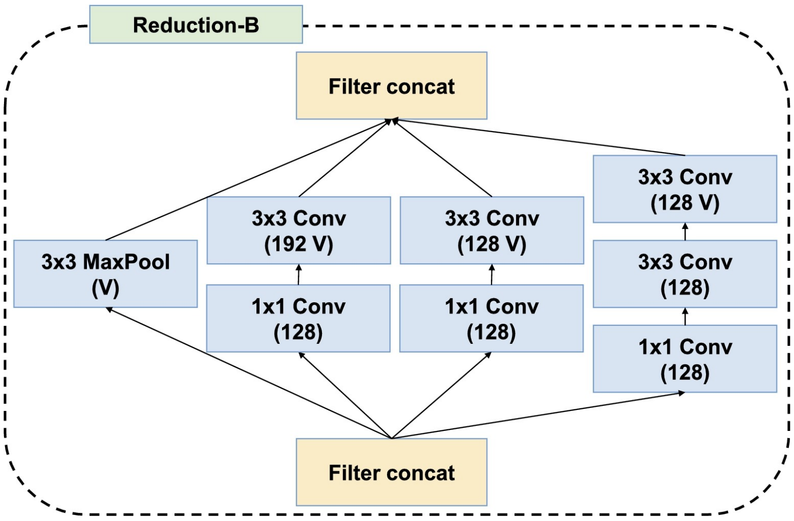


Figure 4.13: SOTA-CNN Reduction-B block

4.3.4 SOTA-LSTM

As mentioned earlier, each SOTA-LSTM model is trained on messages of a single ID. Each of the SOTA-LSTM models has the following architecture and parameter settings. The models first have two dense layers with 128 neurons each and the hyperbolic tangent as activation function. These layers are followed by two LSTM layers of 512 neurons each, which also use the hyperbolic tangent as activation function. Afterwards, there is a final dense layer with 64 neurons and the sigmoid activation function.

All the layers described before are alternated with a dropout layer with a dropout rate of 0.2. All layer weights are initialized in the same way as the BL-DNN model, using the uniform glorot initializer.

The models use the Adam optimizer with a learning_rate of 0.001 and the binary crossentropy as the loss function. The model is trained for 10 epochs, but early stopping is implemented if the validation loss does not improve for 3 epochs. A schematic overview of the structure of the model can be seen in Figure 4.14.

After each model is trained on their attack free training sets, the resulting model is again ran on the training set. The losses of all training samples are collected, and the 99th percentile of the losses is determined. This value is then set as the threshold which will be used for future classification. During testing, if a predicted sample acquires a loss higher than the threshold, it is classified as an attack.

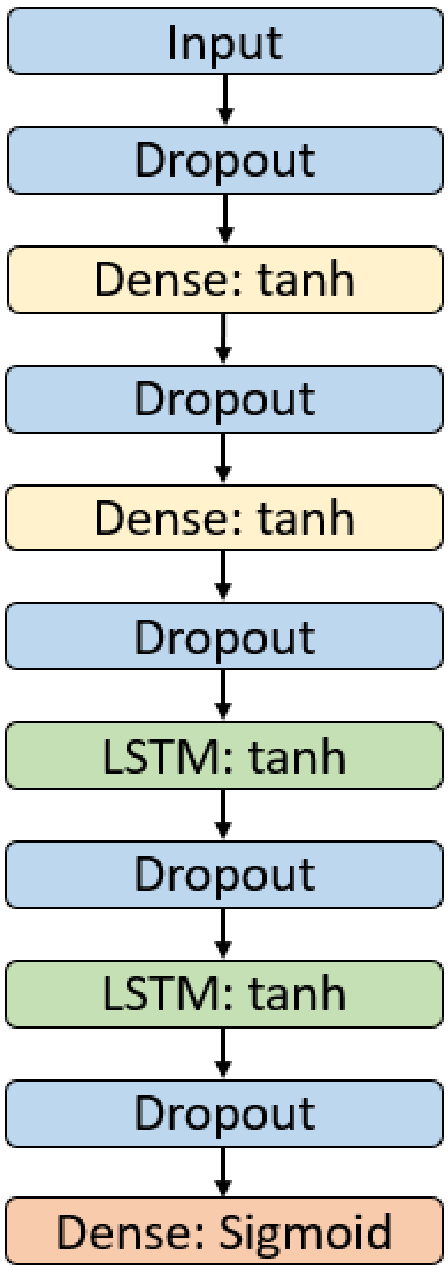


Figure 4.14: SOTA-LSTM schematic

Chapter 5

Results and Analysis

This chapter will show all the results which were retrieved from the previously described tests. This chapter is divided in two sections. The first section covers the results regarding the performance of the models. And the second section will focus on the analysis of the perturbations themselves.

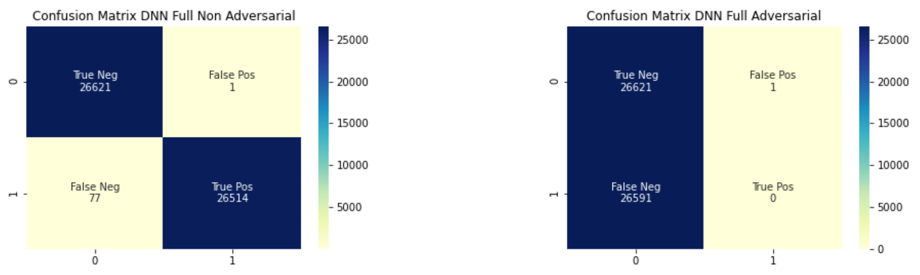
5.1 Model performance

For each model, the previously defined performance metrics will be shown. These performance metrics will be grouped per dataset, showing the difference between the adversarial and the original versions.

5.1.1 BL-DNN

5.1.1.1 Full dataset

Firstly, Figure 5.1 shows heatmaps of the confusion matrix of the BL-DNN model on the Full adversarial dataset (right), and the non perturbed version (left). In the heatmaps, each field is labeled with True Negative, False Negative, True Positive or False Positive. Additionally, the number of sample that belong to these groups is shown. Finally, the color of the heatmap also visualizes the number of samples in each group. The darker the color, the higher the number of samples.



(a) Non adversarial

(b) Adversarial

Figure 5.1: BL-DNN performance on Full dataset with and without perturbations

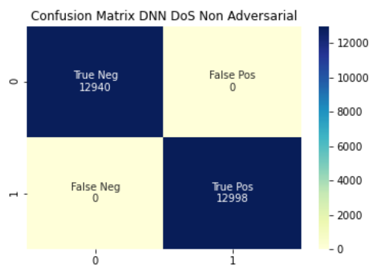
The accompanying metrics can be seen in Table 5.1. From these results, it can be seen that all attack messages are classified as normal messages after adding the perturbations. This results in the accuracy dropping from 0.9985 to 0.5002 The precision, recall and f1-score all drop to 0, since there are no longer any True Positive samples. This means that in all formulas the numerator is set to 0, hence a final value of 0.

	Accuracy	Precision	Recall	f1-score
Non-adversarial	1.0	1.0	1.0	1.0
Adversarial	0.5002	0.0	0.0	0.0

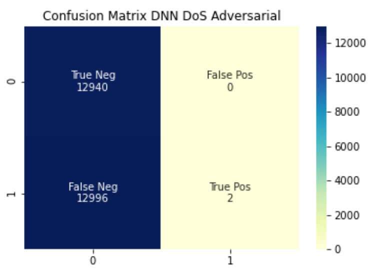
Table 5.1: BL-DNN Performance on Full dataset

5.1.1.2 DoS dataset

The figure and table below contain the metrics of the performance of the BL-DNN model on the DoS dataset. Similar to the Full dataset, nearly all attacks are misclassified once the perturbations are added. This results in a significant drop of accuracy, recall and f1-score.



(a) Non adversarial



(b) Adversarial

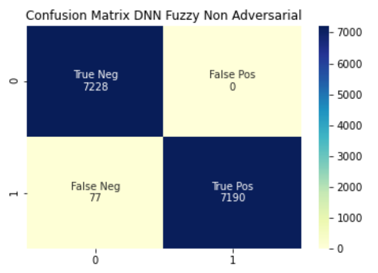
Figure 5.2: BL-DNN performance on DoS dataset with and without perturbations

	Accuracy	Precision	Recall	f1-score
Non-adversarial	1.0	1.0	1.0	1.0
Adversarial	0.4989	1.0	0.0001	0.0003

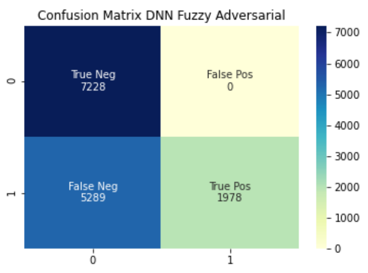
Table 5.2: BL-DNN Performance on DoS dataset

5.1.1.3 Fuzzy dataset

In the Fuzzy dataset case, around 70% of the adversarial samples is misclassified. Once again causing a drop in accuracy, recall and f1-score.



(a) Non adversarial



(b) Adversarial

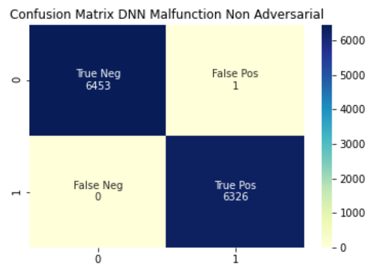
Figure 5.3: BL-DNN performance on Fuzzy dataset with and without perturbations

	Accuracy	Precision	Recall	f1-score
Non-adversarial	0.9946	1.0	09894	0.9946
Adversarial	0.6351	1.0	0.2721	0.4279

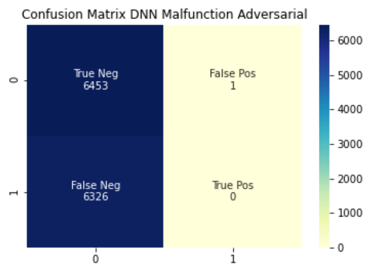
Table 5.3: BL-DNN Performance on Fuzzy dataset

5.1.1.4 Malfunction dataset

Finally, the adversarial samples in the Malfunction dataset are all misclassified. Meaning that the final precision, recall and f1-score are all 0.



(a) Non adversarial



(b) Adversarial

Figure 5.4: BL-DNN performance on Malfunction dataset with and without perturbations

	Accuracy	Precision	Recall	f1-score
Non-adversarial	0.9999	0.9998	1.0	0.9999
Adversarial	0.5049	0.0	0.0	0.0

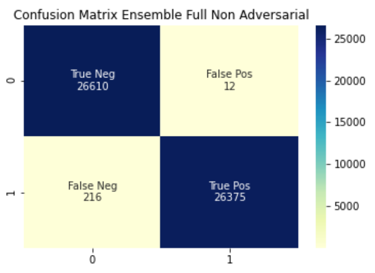
Table 5.4: BL-DNN Performance on Malfunction dataset

5.1.2 BL-Ensemble

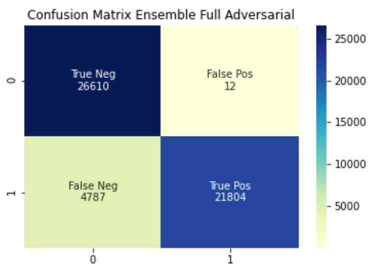
The results below show the performance of the BL-Ensemble model on the adversarial datasets. Note that since the adversarial samples were generated on the BL-DNN model, any change in performance is caused by the transferability of the adversarial samples.

5.1.2.1 Full dataset

In the full adversarial dataset, 20% of the adversarial samples is misclassified. While this is significantly lower percentage than on the BL-DNN model, 1 out of 5 attacks is still able to evade the IDS.



(a) Non adversarial



(b) Adversarial

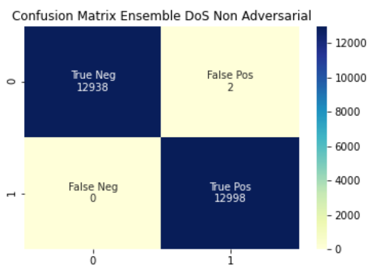
Figure 5.5: BL-Ensemble performance on Full dataset with and without perturbations

	Accuracy	Precision	Recall	f1-score
Non-adversarial	0.9957	0.9995	0.9918	0.9956
Adversarial	0.9098	0.9994	0.8199	0.9008

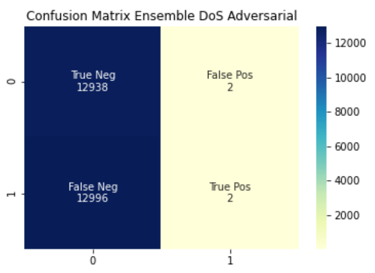
Table 5.5: BL-Ensemble Performance on Full dataset

5.1.2.2 DoS dataset

In contrast to the Full adversarial dataset, nearly all adversarial samples in the DoS dataset are misclassified as can be seen in Figure 5.6.



(a) Non adversarial



(b) Adversarial

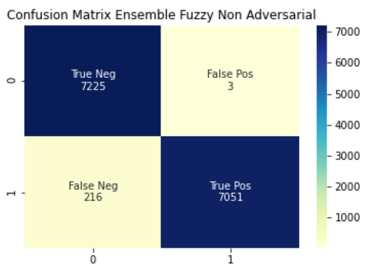
Figure 5.6: BL-Ensemble performance on DoS dataset with and without perturbations

	Accuracy	Precision	Recall	f1-score
Non-adversarial	0.9999	0.9998	1.0	0.9999
Adversarial	0.4988	0.5	0.0001	0.0003

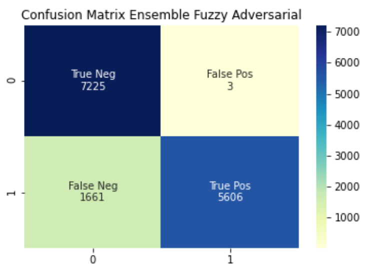
Table 5.6: BL-Ensemble Performance on DoS dataset

5.1.2.3 Fuzzy dataset

The misclassification percentage of the Fuzzy adversarial samples is around 20%, which is similar to the Full dataset.



(a) Non adversarial



(b) Adversarial

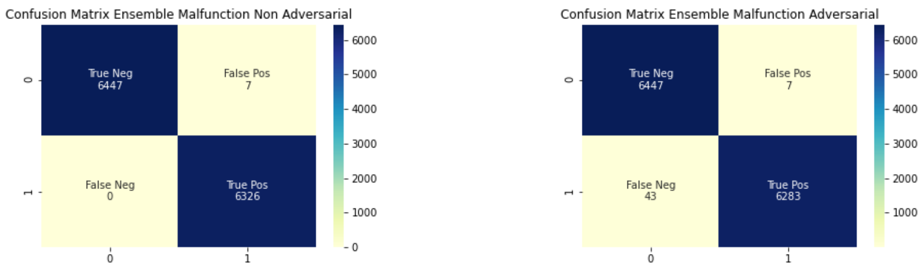
Figure 5.7: BL-Ensemble performance on Fuzzy dataset with and without perturbations

	Accuracy	Precision	Recall	f1-score
Non-adversarial	0.9848	0.9995	0.9702	0.9847
Adversarial	0.8852	0.9994	0.7714	0.8707

Table 5.7: BL-Ensemble Performance on Fuzzy dataset

5.1.2.4 Malfunction dataset

In Figure 5.8 it is visible that nearly none of the adversarial samples causes a misclassification in the BL-Ensemble model.



(a) Non adversarial (b) Adversarial

Figure 5.8: BL-Ensemble performance on Malfunction dataset with and without perturbations

	Accuracy	Precision	Recall	f1-score
Non-adversarial	0.9994	0.9988	1.0	0.9994
Adversarial	0.9960	0.9988	0.9932	0.9960

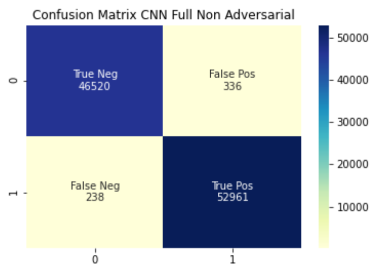
Table 5.8: BL-Ensemble Performance on Malfunction dataset

5.1.3 SOTA-CNN

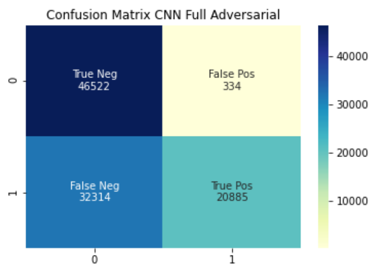
This section will focus on the performance of the SOTA-CNN model. Note that the inputs of the SOTA-CNN only use the message IDs, so if no perturbations are added to the ID, the samples will be equal for the SOTA-CNN model.

5.1.3.1 Full dataset

As can be seen in Figure 5.9 more than half of the adversarial samples generated on the BL-DNN are able to evade the SOTA-CNN model.



(a) Non adversarial



(b) Adversarial

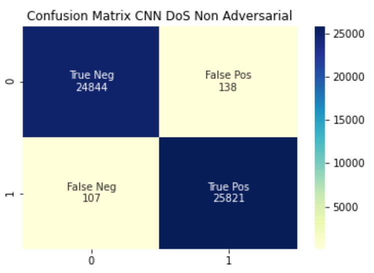
Figure 5.9: SOTA-CNN performance on Full dataset with and without perturbations

	Accuracy	Precision	Recall	f1-score
Non-adversarial	0.9942	0.9936	0.9955	0.9946
Adversarial	0.6736	0.9842	0.3925	0.5612

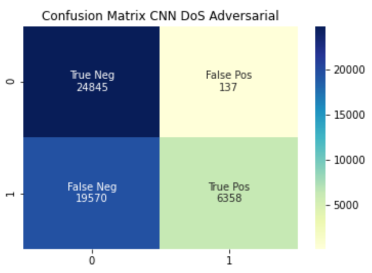
Table 5.9: SOTA-CNN Performance on Full dataset

5.1.3.2 DoS dataset

When generating the adversarial samples for the DoS attacks, the message ID was limited to a maximum value of 7. Specifically, only the final 3 bits of the message ID were included in the perturbation generation. Even under this restriction, most of the adversarial samples are misclassified by the SOTA-CNN model.



(a) Non adversarial



(b) Adversarial

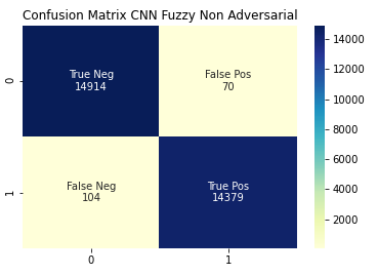
Figure 5.10: SOTA-CNN performance on DoS dataset with and without perturbations

	Accuracy	Precision	Recall	f1-score
Non-adversarial	0.9951	0.9946	0.9958	0.9952
Adversarial	0.6129	0.9789	0.2452	0.3921

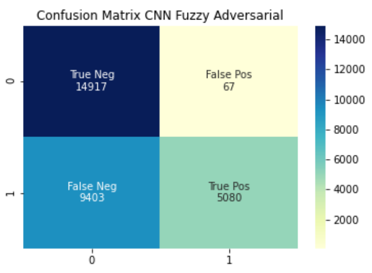
Table 5.10: SOTA-CNN Performance on DoS dataset

5.1.3.3 Fuzzy dataset

The SOTA-CNN model misclassifies around 60% of the adversarial Fuzzy samples. This results in a drop of the recall from 0.9928 to 0.3507.



(a) Non adversarial



(b) Adversarial

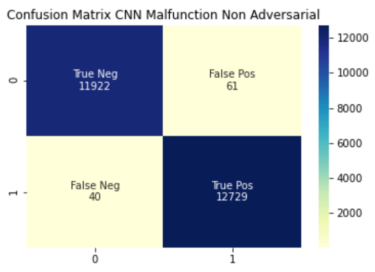
Figure 5.11: SOTA-CNN performance on Fuzzy dataset with and without perturbations

	Accuracy	Precision	Recall	f1-score
Non-adversarial	0.9940	0.9951	0.9928	0.9939
Adversarial	0.6786	0.9869	0.3507	0.5175

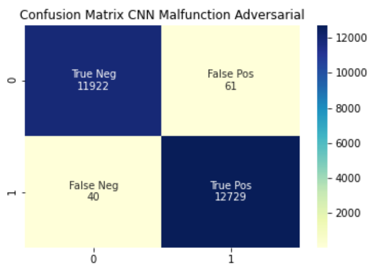
Table 5.11: SOTA-CNN Performance on Fuzzy dataset

5.1.3.4 Malfunction dataset

In the Malfunction dataset, there is no difference between the dataset with or without the adversarial samples. The reason behind this is that the message ID was not perturbed during the adversarial sample generation. This means that all perturbations occur in features that are not relevant to the SOTA-CNN.



(a) Non adversarial



(b) Adversarial

Figure 5.12: SOTA-CNN performance on Malfunction dataset with and without perturbations

	Accuracy	Precision	Recall	f1-score
Non-adversarial	0.9959	0.9952	0.9968	0.9960
Adversarial	0.9959	0.9952	0.9968	0.9960

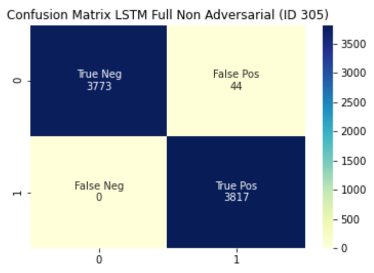
Table 5.12: SOTA-CNN Performance on Malfunction dataset

5.1.4 SOTA-LSTM

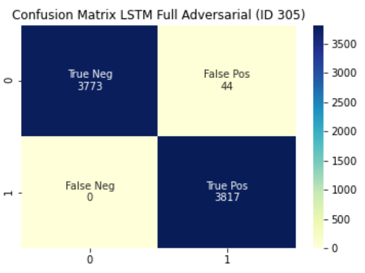
Since there are several SOTA-LSTM models, the decision was made to show the confusion matrices of only one of the models. Whilst the other metrics will be computed by averaging the metrics of each model. The confusion matrices are all from the SOTA-LSTM corresponding to message ID 305.

5.1.4.1 Full dataset

It can be seen from both Figure 5.13 and Table 5.13 that the adversarial samples have no influence on the models. The average metrics remain the same, and so do the two confusion matrices.



(a) Non adversarial



(b) Adversarial

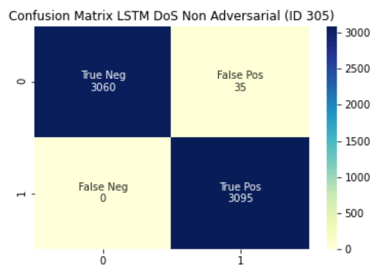
Figure 5.13: SOTA-LSTM performance on Full dataset with and without perturbations

	Accuracy	Precision	Recall	f1-score
Non-adversarial	0.9491	0.9925	0.9043	0.9341
Adversarial	0.9491	0.9925	0.9044	0.9342

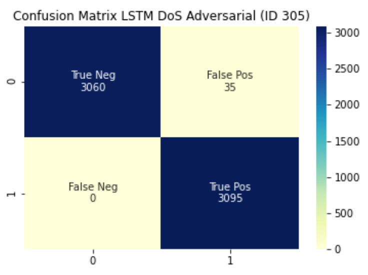
Table 5.13: SOTA-LSTM Performance on Full dataset

5.1.4.2 DoS dataset

While there are no differences between the two confusion matrices, all the performance metrics increase in the presence of the adversarial samples. Since these are average metrics, this indicates that on average, the SOTA-LSTM models are better at detecting the adversarial samples than the original attack messages.



(a) Non adversarial



(b) Adversarial

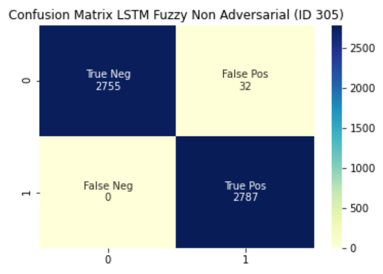
Figure 5.14: SOTA-LSTM performance on DoS dataset with and without perturbations

	Accuracy	Precision	Recall	f1-score
Non-adversarial	0.8980	0.7977	0.8	0.7988
Adversarial	0.9980	0.9961	1.0	0.9980

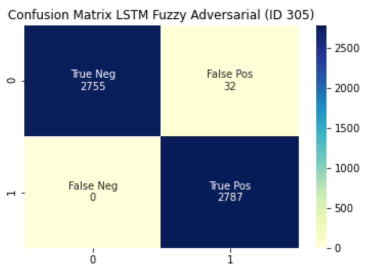
Table 5.14: SOTA-LSTM Performance on DoS dataset

5.1.4.3 Fuzzy dataset

Similarly to the Full dataset, there are no noticeable differences regarding the performance of the SOTA-LSTM models with or without the adversarial samples.



(a) Non adversarial



(b) Adversarial

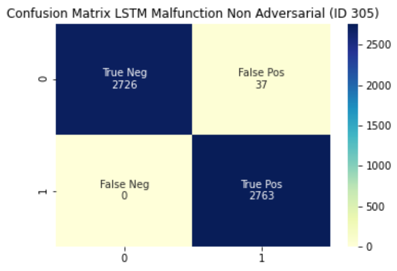
Figure 5.15: SOTA-LSTM performance on Fuzzy dataset with and without perturbations

	Accuracy	Precision	Recall	f1-score
Non-adversarial	0.9979	0.9960	1.0	0.9979
Adversarial	0.9979	0.9960	1.0	0.9979

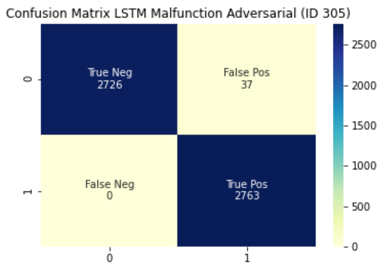
Table 5.15: SOTA-LSTM Performance on Fuzzy dataset

5.1.4.4 Malfunction dataset

Finally, just as with the Full and Fuzzy datasets, there are no changes in the performance metrics.



(a) Non adversarial



(b) Adversarial

Figure 5.16: SOTA-LSTM performance on Malfunction dataset with and without perturbations

	Accuracy	Precision	Recall	f1-score
Non-adversarial	0.9965	0.9931	1.0	0.9965
Adversarial	0.9965	0.9931	1.0	0.9965

Table 5.16: SOTA-LSTM Performance on Malfunction dataset

5.1.5 Performance Analysis

The drop in f1-score for each of the models on each dataset, has been plotted as a bar chart in Figure 5.17. Note that since the drop in f1-score is plotted, the larger the bar, the worse the model performed on the adversarial samples.

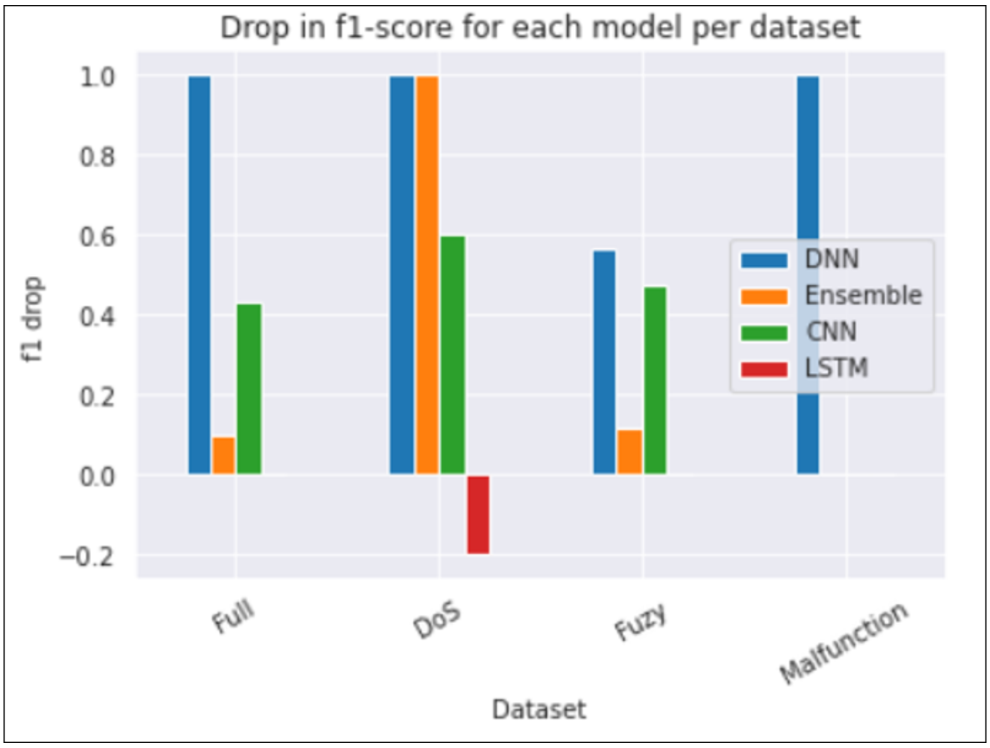


Figure 5.17: Bar plot of f1-score drop

That the adversarial samples can generate misclassifications is very clear to see in the performance metrics of the BL-DNN model. However, since the samples were generated on this model, this is only to be expected. These adversarial samples have also shown to be transferable from the BL-DNN model to other models. Especially the DoS adversarial samples caused many misclassifications in both the BL-Ensemble and the SOTA-CNN model, as can be seen from the large DoS bars in Figure 5.17.

However, not all models were equally vulnerable to the adversarial samples. The SOTA-LSTM models showed no loss in performance when confronted with the adversarial samples. In the presence of the presence of the DoS adversarial samples, the performance of the models even increased, denoted by a negative bar in Figure 5.17. The current hypothesis for this is that the attack method is not optimized for recurrent networks. This will be expaneded upon in Chapter 6.

5.2 Perturbation Analysis

In this section, a close look wil be taken into the exact perturbations that caused the misclassifications. First, the number of iterations and the size of the perturbations will be reviewed. Afterwards, the specific features that were perturbed will be highlighted.

5.2.1 Iterations and Size

Let's first look at the statistics of the perturbation size and the number of iterations for the Full dataset. The statistics contain the following metrics. First the count, which signals the number of data points used for the statistics. Next the mean and std, which state the mean value and the standard deviation of the data. Additionally, the min and max values are included, which state the minimum and maximum values found in the data. Finally, 25%, 50% and 75% denote the 25th, 50th and 75th percentile of the data respectively.

	Perturbation Size	**Iterations**
count	26591	26591
mean	4.212	4.582
std	3.891	4.033
min	0.0	0.0
25%	2.0	2.0
50%	2.011	3.0
75%	4.002	5.0
max	35.327	36.0

Table 5.17: Statistics of perturbation size and iterations for Full dataset

The table shows that the majority of the adversarial samples were computed within 5 iterations, with an average perturbation size of around 4. There was at least one outlier that required 36 iterations to converge, but there were no samples that failed to find an adversarial perturbation. Otherwise, the maximum of the iterations would have been 50 (as it was defined in the parameters during generation of the samples).

Next, in Table 5.18, the statistics of the iterations and perturbation size with and without the failed samples can be seen. It can be seen that the majority of the samples is generated in 2 iterations with and matching perturbation size of 2. However, there are two samples that fail to reach a successful perturbation in 50 iterations. Since the maximum perturbation size is quite low at 4, it can be deduced that these samples most likely reached a loop where a certain feature was continuously increased and then decreased again. Thus failing to converge.

	Perturbation Size	**Iterations**
count	12998	12998
mean	2.0	2.007
std	0.024	0.595
min	2.0	2.0
25%	2.0	2.0
50%	2.0	2.0
75%	2.0	2.0
max	4.0	50.0

(a) DoS perturbation statistics with failed samples

	Perturbation Size	**Iterations**
count	12996	12996
mean	2.0	2.0
std	0.0	0.0
min	2.0	2.0
25%	2.0	2.0
50%	2.0	2.0
75%	2.0	2.0
max	2.0	2.0

(b) DoS perturbation statistics without failed samples

Table 5.18: DoS perturbation statistics with and without failed samples

For the Fuzzy dataset, Table 5.19 also shows the relevant statistics with and without the failed samples. The average perturbation size is around 16 within 19 iterations. Additionally, there were nearly 2000 samples which failed to converge within 50 iterations. By removing these samples from the statistics, the new average perturbation size lies around 8 within 8 iterations. Note that the maximum perturbation size still reaches 48, meaning that the majority of the features has been perturbed.

	Perturbation Size	**Iterations**
count	7267	7267
mean	15.867	19.697
std	13.344	19.070
min	0.0	0.0
25%	6.0	6.0
50%	10.0	10.0
75%	30.0	50.0
max	50.0	50.0

(a) Fuzzy perturbation statistics with failed samples

	Perturbation Size	**Iterations**
count	5289	5289
mean	8.355	8.364
std	5.216	5.269
min	0.0	0.0
25%	5.0	5.0
50%	7.0	7.0
75%	11.0	11.0
max	48.0	49.0

(b) Fuzzy perturbation statistics without failed samples

Table 5.19: Fuzzy perturbation statistics with and without failed samples

Finally, for the Malfunction dataset, the statistics are shown in Table 5.20. It can be seen that all the samples reach the succesfull perturbation in 4 iterations with a perturbation size of 4.

	Perturbation Size	**Iterations**
count	6326	6326
mean	4.0	4.0
std	0.0	0.0
min	4.0	4.0
25%	4.0	4.0
50%	4.0	4.0
75%	4.0	4.0
max	4.0	4.0

Table 5.20: Malfunction perturbation statistics

So for all datasets, if the adversarial sample converges, they do so in a relatively small number of iterations. The majority of the samples was able to reach their goal within 8 iterations. This shows that the changes do not have to be large to cause a misclassification.

5.2.2 Feature Analysis

Now, a closer look will be taken to what features played the largest role in the perturbations. Figure 5.18 shows a heatmap where for each adversarial dataset, the importance of each input feature is shown.

In the heatmap various interesting patterns are visible. Firstly, for both the Full and the Fuzzy dataset, the spread of the features is the widest. However, the Full dataset seems to put a larger priority on several message ID bits.

In contrast to the Full and Fuzzy dataset, both the DoS and the Malfunction dataset only perturb a limited number of features. For the DoS dataset, only two features ever get perturbed. These features are the 10th message ID bit and the 4th data bit. The Malfunction dataset seems to only perturb a set of 7 features. The most perturbed being the 4th data bit.

In general, the lower message ID bits seem quite influential in the perturbations. This could signal that the models highly depend on these bits, especially when you look at the performance of the DoS datast. The 4th data bit additionally plays a large role both in the DoS and the Malfunction dataset. Finally, there seems to be a small hotspot around the first four bits of the 7th data byte. These bits are more relevant for the Full and Fuzzy datasets. Doing further feature analysis is outside the scope of the thesis, so currently no reason can be provided as to why these features are more relevant.

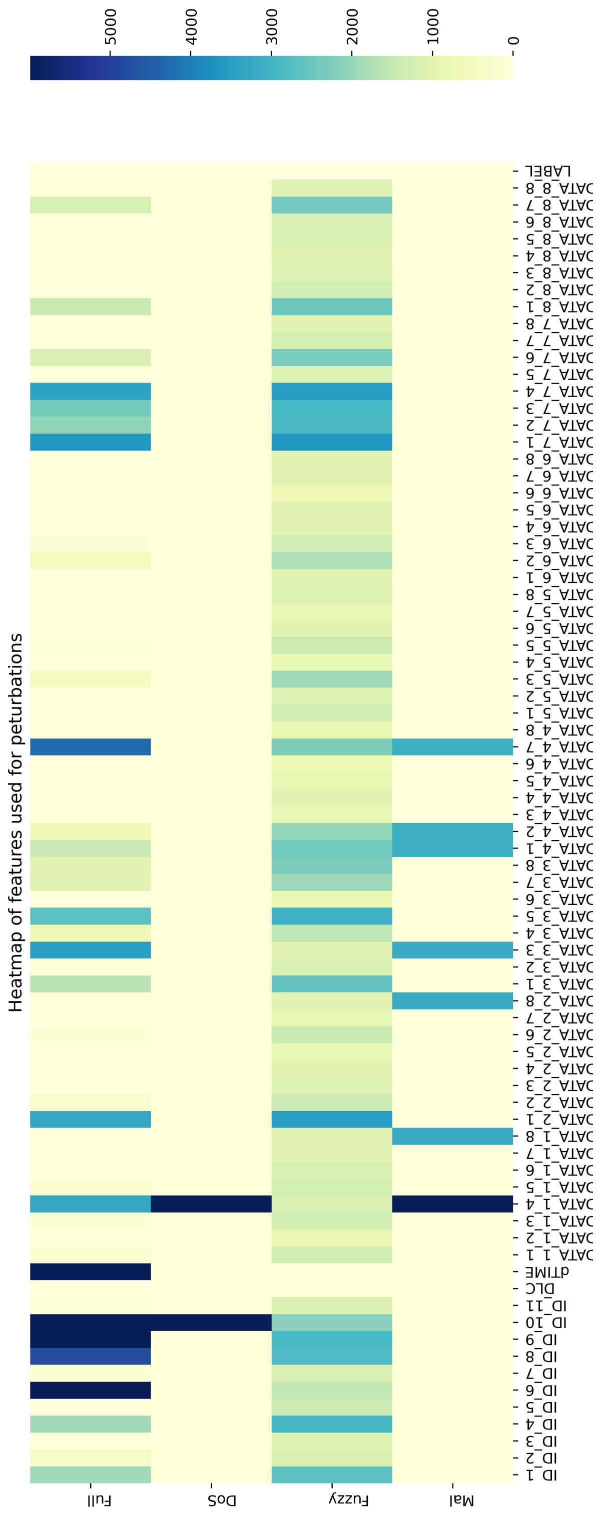


Figure 5.18: Heatmap of the perturbation features

Chapter 6

Discussion

The original research question of the thesis was two-fold. What are the weaknesses of DL techniques that are present in DL-based IDSs for CAN? Can these weaknesses influence the performance of the IDSs? From the results it can be seen that the adversarial samples caused a decrease in f1-score of on average 89% on the BL-DNN, 30% on the BL-Ensemble and 38% on the SOTA-CNN model. The drop in performance between the datasets without and with the perturbations can be used to determine the weakness against perturbations for the different models. Additionally, since the perturbations were generated on only the BL-DNN model, the performance drop on the other models can show the presence of the transferability weakness in the models.

Since the SOTA-CNN model solely looks at the message ID bits, not all parts of the perturbation are of significance for this model. This means that the performance of the SOTA-CNN could potentially be further degraded by generating the various adversarial datasets purely on the message ID bits. Additionally, the experiments are set up to show the transferability of the adversarial samples. This means that these samples are not generated on and optimized for the BL-Ensemble and SOTA-CNN models. The performance of these models could be further degraded by generating the adversarial samples directly on them.

Regarding the SOTA-LSTM model, none of the adversarial samples influenced the performance of the models in a negative way. The most likely reason for this lies in the nature of RNNs and LSTMs. While other forms of neural networks learn information and correlations within single inputs, these recurrent networks additionally learn correlations between inputs. The FGSM generates perturbations to alter the information within an input, but this does not change the correlations between messages. Therefore there is still enough information present in the correlations for the SOTA-LSTM to correctly classify the adversarial samples. To further research the resilience of these types of models, the attack method has to be adapted to handle several consecutive messages. This will allow the attack method to alter the correlations between the messages as well.

Finally, while the important features for the perturbations could be identified and visualized, no reason behind why these features are important was found. This is mainly since doing further feature analysis was outside of the scope of the thesis. This means that no analysis was done to retrieve information that was not immediately visible. For example, the encoding that vehicle manufacturers use are company secrets. This means that it is unknown what the data bytes translate to, so no further information can be gained from this. Additionally, what message IDs correspond to what ECUs/vehicle functions is another company secret. Although the importance of these features may lie in their decoded meaning, it is impossible to research without the decoding information or extensive testing on a physical vehicle.

Chapter 7

Conclusions and Future work

This final chapter will wrap up the thesis by providing my insights into the results and the process of the project. First, the conclusions following the results will be given as well as their implications. Next, the limitations of the results and the thesis as a whole will be given. This is to provide further context to the results and conclusions. After this, several suggestions are given for potential future work. These suggestions will range from overcoming the previously described limitations to other work building upon the current findings. The final section provides my reflections on the results and thesis.

7.1 Conclusions

The aim of this thesis was to investigate the resilience of DL-based IDSs for automotive networks. Specifically the resilience against adversarial samples resulting from small perturbations. Additionally, the transferability of these perturbations were explored.

The results show that the adversarial samples have a large impact on the BL-DNN model. The recall and f1-score on all datasets drop by at least 57% and for most datasets the drop is more than 90%. Since the adversarial samples were generated on the BL-DNN model, this does not yet show the transferability.

The Full, DoS and Fuzzy datasets were all able to influence the performance of the BL-Ensemble model and even the SOTA-CNN model. The f1-score on the SOTA-CNN model decreased by at most 60%. Thus showing the vulnerability of the state-of-the-art model to perturbations, and the transferability of these perturbations. However, the perturbations were not transferable to all models. The SOTA-LSTM models remained mainly unaffected. In the presence of the DoS adversarial samples, the performance of the SOTA-LSTM models even increased. This shows that the combination of model architecture and attack type might even counteract the transferability of the perturbations.

To conclude, DL techniques are vulnerable to adversarial samples created using small perturbations. This weakness is also present in certain state-of-the-art DL-based IDSs for vehicle networks. Additionally, these perturbations can be generated on one model and then transferred to another model. This opens the possibility of an attacker creating their own model to generate attacks on. After which these attacks can be transferred to their desired target model. It is therefore imperative that this vulnerability is taken into account by researchers when creating future IDSs.

7.2 Limitations

The first limitation is that of the results of the SOTA-LSTM models. As shown in the results, these models are unaffected by the perturbations that were generated. A hypothesis was provided as to why this could be the case, however this hypothesis remains untested. So without further research, the conclusion regarding the resilience of the SOTA-LSTM model towards perturbations is still preliminary.

Another one of the main limitations is that the results are limited to only two state-of-the-art IDSs. There have been many more research papers that presented their own DL-based IDSs using different and sometimes more advanced techniques. However, time constraints and the complexity of some of these models prevented me re-creating more models. So the results are limited to the models presented in the thesis.

Additionally, when creating the adversarial datasets, the attack messages were altered in such a way that they had the highest chance to still be functional. However, since no physical test-bed was available, be it in the form of several ECUs or an actual vehicle, there was no way to actually test whether the adversarial attack messages are still functional or not.

Finally, all the adversarial samples were generated using the FGSM. However, there are various other methods which are also capable of computing adversarial samples. These methods include the Jacobian Saliency-map Attack and DeepFool amongst others. It would have been interesting to see the differences between the adversarial samples generated by the various methods. This was left for future research due to time constraints.

7.3 Future work

The future work can be divided into two general sections. First, the future work that covers the limitations of the current thesis. Second, the future work that builds upon the results from this thesis

7.3.1 Fixing limitations

As mentioned in the previous section, the current thesis project is still quite limited in various aspect. This obviously opens the path for future research to focus on these limitations

7.3.1.1 Adapting to recurrent models

As shown in the results, the perturbations were unable to negatively affect the recurrent networks, i.e. the SOTA-LSTM models. Future research could improve on these results by adapting the FGSM to additionally perturb the information in a sequence of messages instead of only in a single message. This would allow for further testing whether the recurrent networks are resilient against perturbations in general, or just the perturbations generated on individual messages.

7.3.1.2 Other IDS models

Currently, there are only four models which are tested, of which only two are state-of-the-art. Future research could include other models such as CANnolo [58] and CANIntelliIDS [63]. CANnolo works quite similarly to the RNN model used in this thesis, where an LSTM-based model is used to re-create a message, and based on the reconstruction error a classification is made. However, while the model in the thesis reconstructs the next message based on the input, CANnolo reconstructs the input message using and autoencoder with LSTM layers.

CANIntelliIDS uses a combination of CNN and RNN type layers. This allows it to focus on the flow of the data in the messages without having to create a separate LSTM model per message ID. Additionally, they use Gated Recurrent Units (GRU) for their RNN layers in combination with the attention mechanism. This is a mechanism that was introduced for natural language processing, which allows a model to give additional contextual meaning to the different inputs. This means that the model is capable of gaining more information from the location of the inputs with respect to other inputs.

These are just some of examples of other DL-based IDSs for CAN, and there are many more. Each of these have their own unique qualities which may or may not help them to overcome adversarial samples.

7.3.1.3 Other attack methods

This thesis focused solely on the Fast Gradient-Sign method to generate the adversarial samples. However, there are many other methods that use different approaches to achieve the same goal. For example, the Jacobian-based Saliency Map Attack [29] and DeepFool [30]. Future research can investigate whether the different methods have different effects regarding performance and transferability on DL-based IDSs for CAN.

7.3.1.4 Different test environments

One of the main intentions of this thesis during the creation of the adversarial samples is to make sure that the attacks still have the highest chance of being functional. In the current test environment, there was no way to actually confirm this, reducing the weight of the results. This could be solved by performing similar tests on either a prototype test-bed or preferably on an actual vehicle. This would allow the researchers to confirm that not only the attacks would avoid the IDS but would also still have the expected effect on the vehicle itself.

7.3.2 Future research

The following step after showing the presence of the weaknesses is to solve these weaknesses. Future research could focus on know techniques such as distillation for defense or re-training on adversarial datasets. Different model architectures could also result in a better resilience against adversarial samples. Generally speaking, autoencoder networks or DBNs have a higher tolerance to small errors, which could make them a good choice to incorporate in the new models. An architecture that combines the physical and digital features of the CAN messages and ECUs could also improve current performance. This model could potentially detect more intricate attacks as well as determine the sender of these attacks. This combination of features could increase the resilience against adversarial samples.

7.4 Reflections

As with any thesis, it is important to reflect on what impact the work has on various aspects of society. These aspects are generally economic, social, environmental and ethical. This section will cover the economic, social and ethical consequences of the thesis. The environmental aspect will not be reviewed, since this project has no direct influence on this aspect.

For the economical aspect, the influence will be visible mainly for vehicle manufacturers. By improving on the weaknesses shown in this thesis, they can make their vehicles more secure against future attackers. This in turn allows them to prevent possible ransom demands from attackers or damage claims from potential victims. The owners of the vehicles are not able to directly apply the findings of this thesis themselves, but they still might encounter the economic influence. If the security of a vehicle is improved by the manufacturers, the chance of damages or even theft of the vehicle will be lowered. This will be economically beneficial for the vehicle owners.

The social aspect of this thesis relies mainly on improving the safety of modern vehicles. Now that the weaknesses in the DL-based IDSs are known, they can be overcome. By detecting evasion attempts on the systems, actions can be taken to prevent the attackers from gaining control of the system. This will make the systems within the vehicles more secure, and thus a higher degree of safety can be guaranteed for the drivers, passengers and other road users or pedestrians.

Finally, the ethical aspect of this thesis revolves around the implications of revealing the presence of these weaknesses. On the one hand, revealing these weaknesses is a very important step. If they are not revealed, and are instead discovered and exploited by attackers in secret, the consequences could be dire. By revealing them publicly, awareness for these issues can be raised, thus increasing the research to overcome these weaknesses.

On the other hand, by revealing these weaknesses publicly, potential attackers will also be notified. This could lead them to exploit these weaknesses using similar methods described in the thesis. This however is quite unlikely. As mentioned previously, in order for an attacker to send messages on the CAN bus and evade the IDS, they first have to get access to this system. This in itself is already not an easy task. Next, every vehicle manufacturer treats their security measures as company secrets. What type of IDS are used is unknown, and the same holds for what features they use or how they are implemented. The likelihood that current vehicle security systems depend on DL techniques is low due to several factors such as the novelty of the technology.

Overall, the advantages outweigh the disadvantages regarding the revelation of these weaknesses. The goal of this thesis is to stimulate future researchers to improve on the DL models and thus increase the security of the vehicles.

References

[1] H. M. Song, J. Woo, and H. K. Kim, "In-vehicle network intrusion detection using deep convolutional neural network," *Vehicular Communications*, vol. 21, p. 100198, 2020. doi: https://doi.org/10.1016/j.vehcom.2019.100198. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214209619302451 [Pages i, 27, 39, 43, and 61.]

[2] R. N. Charette, "This car runs on code," *IEEE spectrum*, vol. 46, no. 3, p. 3, 2009. [Online]. Available: https://spectrum.ieee.org/this-car-runs-on-code [Page 1.]

[3] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," *20th USENIX Security Symposium (USENIX Security 11)*, 2011. doi: 10.1109/TCCN.2018.2835460. [Online]. Available: https://www.usenix.org/legacy/event/sec11/tech/full_papers/Checkoway.pdf [Pages 1 and 11.]

[4] Bosch, "Can specification 2.0," 1991. [Online]. Available: www.can-cia.de/fileadmin/cia/specifications/can20a.pdf [Pages 1 and 8.]

[5] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *2010 IEEE Symposium on Security and Privacy*, 2010. doi: 10.1109/SP.2010.34 pp. 447–462. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5504804?casa_token=4s1pq125b20AAAAA:EO5fuqkSTnsJYeqB17yRq25gKkljd6mh6Q8_yrlg6a97q5jqCLeGgA6dOlmlWqEK-YOofZdf [Pages 1, 10, and 38.]

[6] D. K. Nilsson, U. E. Larson, and E. Jonsson, “Efficient in-vehicle delayed data authentication based on compound message authentication codes,” in *2008 IEEE 68th Vehicular Technology Conference*, 2008. doi: 10.1109/VETECF.2008.259 pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4657091?casa_token=XMYkj_64TY4AAAAA:myUT5LA_oyANi6JkmnEN1rHS6qlJ5G_zfopYuFPnEKXAQGN7gXUXsQOsHkFK6Gt7PiC7eOUt [Pages 1 and 11.]

[7] M. Wolf, A. Weimerskirch, and T. Wollinger, “State of the art: Embedding security in vehicles,” *EURASIP Journal on Embedded Systems*, vol. 2007, pp. 1–16, 2007. [Online]. Available: https://link.springer.com/content/pdf/10.1155/2007/74706.pdf [Pages 2 and 11.]

[8] A. L. Buczak and E. Guven, “A survey of data mining and machine learning methods for cyber security intrusion detection,” *IEEE Communications Surveys Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016. doi: 10.1109/COMST.2015.2494502. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7307098 [Pages 2, 22, and 23.]

[9] R. Chalapathy and S. Chawla, “Deep learning for anomaly detection: A survey,” *CoRR*, vol. abs/1901.03407, 2019. [Online]. Available: http://arxiv.org/abs/1901.03407 [Pages 2 and 21.]

[10] M. Wolf, A. Weimerskirch, and C. Paar, “Security in automotive bus systems,” in *Workshop on Embedded Security in Cars*. Citeseer, 2004, pp. 1–13. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.728&rep=rep1&type=pdf [Pages 7, 8, and 11.]

[11] N. Weiss, “Automotive-specific documentation.” [Online]. Available: https://scapy.readthedocs.io/en/latest/layers/automotive.html [Pages vii and 8.]

[12] D. K. Nilsson, P. H. Phung, and U. E. Larson, “Vehicle ecu classification based on safety-security characteristics,” in *IET Road Transport Information and Control - RTIC 2008 and ITS United Kingdom Members’ Conference*, 2008. doi: 10.1049/ic.2008.0810 pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4562233 [Page 10.]

[13] T. Hoppe, S. Kiltz, and J. Dittmann, “Security threats to automotive can networks – practical examples and selected short-term countermeasures,” *Reliability Engineering System Safety*, vol. 96, no. 1, pp. 11–25, 2011. doi: https://doi.org/10.1016/j.ress.2010.06.026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0951832010001602 [Page 10.]

[14] U. E. Larson and D. K. Nilsson, “Securing vehicles against cyber attacks,” in *Proceedings of the 4th Annual Workshop on Cyber Security and Information Intelligence Research: Developing Strategies to Meet the Cyber Security and Information Intelligence Challenges Ahead*, ser. CSIIRW ’08. New York, NY, USA: Association for Computing Machinery, 2008. doi: 10.1145/1413140.1413174. ISBN 9781605580982. [Online]. Available: https://doi.org/10.1145/1413140.1413174 [Page 11.]

[15] U. E. Larson, D. K. Nilsson, and E. Jonsson, “An approach to specification-based attack detection for in-vehicle networks,” in *2008 IEEE Intelligent Vehicles Symposium*, 2008. doi: 10.1109/IVS.2008.4621263 pp. 220–225. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4621263?casa_token=0RP1BNl7l_IAAAAA:QDrTuq1VnaYHi8gIcJZ13cUDlLLD3vOeeNKoLJWATIG5GAcO26RX3CirGzSb6hixw37IBZzZ [Page 11.]

[16] T. Hoppe, S. Kiltz, and J. Dittmann, “Applying intrusion detection to automotive it-early insights and remaining challenges,” *Journal of Information Assurance and Security (JIAS)*, vol. 4, no. 6, pp. 226–235, 2009. [Online]. Available: https://www.researchgate.net/profile/Tobias-Hoppe-2/publication/285312982_Applying_intrusion_detection_to_automotive_IT-early_insights_and_remaining_challenges/links/56f41b9e08ae81582bf09f50/Applying-intrusion-detection-to-automotive-IT-early-insights-and-remaining-challenges.pdf [Page 12.]

[17] D. G. Kleinbaum, K. Dietz, M. Gail, M. Klein, and M. Klein, *Logistic regression*. Springer, 2002. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4419-1742-3? [Page 12.]

[18] Y.-Y. Song and L. Ying, “Decision tree methods: applications for classification and prediction,” *Shanghai archives of psychiatry*, vol. 27, no. 2, p. 130, 2015. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4466856/ [Page 13.]

[19] S. Suthaharan, "Support vector machine," in *Machine learning models and algorithms for big data classification*. Springer, 2016, pp. 207–235. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4899-7641-3_9 [Page 13.]

[20] L. E. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009. [Online]. Available: http://scholarpedia.org/article/K-nearest_neighbor [Page 14.]

[21] K. P. Murphy *et al.*, "Naive bayes classifiers," *University of British Columbia*, vol. 18, no. 60, pp. 1–8, 2006. [Online]. Available: https://www.ic.unicamp.br/~rocha/teaching/2011s1/mc906/aulas/naive-bayes.pdf [Page 14.]

[22] C. C. Aggarwal *et al.*, "Neural networks and deep learning," *Springer*, vol. 10, pp. 978–3, 2018. [Online]. Available: https://link.springer.com/book/10.1007/978-3-319-94463-0?noAccess=true [Page 14.]

[23] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *towards data science*, vol. 6, no. 12, pp. 310–316, 2017. [Online]. Available: https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf [Page 14.]

[24] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*. Ieee, 2017, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8308186 [Page 15.]

[25] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019. [Online]. Available: https://direct.mit.edu/neco/article-abstract/31/7/1235/8500/A-Review-of-Recurrent-Neural-Networks-LSTM-Cells [Page 15.]

[26] I. Corona, G. Giacinto, and F. Roli, "Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues," *Information Sciences*, vol. 239, pp. 201–225, 2013. doi: https://doi.org/10.1016/j.ins.2013.03.022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025513002119 [Pages 16 and 29.]

[27] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” 2014. [Online]. Available: https://arxiv.org/abs/1312.6199 [Pages 16, 17, and 18.]

[28] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014. [Online]. Available: https://arxiv.org/abs/1412.6572 [Pages vii, 16, 17, and 41.]

[29] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, “The limitations of deep learning in adversarial settings,” in *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, 2016. doi: 10.1109/EuroSP.2016.36 pp. 372–387. [Online]. Available: https://ieeexplore.ieee.org/document/7467366 [Pages 17, 19, and 94.]

[30] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “Deepfool: a simple and accurate method to fool deep neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2574–2582. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2016/html/Moosavi-Dezfooli_DeepFool_A_Simple_CVPR_2016_paper.html [Pages 17, 19, and 94.]

[31] G. Zizzo, C. Hankin, S. Maffeis, and K. Jones, “Invited: Adversarial machine learning beyond the image domain,” in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8806924 [Page 17.]

[32] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. D. McDaniel, “Adversarial perturbations against deep neural networks for malware classification,” *CoRR*, vol. abs/1606.04435, 2016. [Online]. Available: http://arxiv.org/abs/1606.04435 [Pages 17 and 19.]

[33] N. Papernot, P. D. McDaniel, and I. J. Goodfellow, “Transferability in machine learning: from phenomena to black-box attacks using adversarial samples,” *CoRR*, vol. abs/1605.07277, 2016. [Online]. Available: http://arxiv.org/abs/1605.07277 [Pages 18 and 37.]

[34] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, “Distillation as a defense to adversarial perturbations against deep neural networks,” in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.

doi: 10.1109/SP.2016.41 pp. 582–597. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7546524 [Pages 19 and 20.]

[35] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017. doi: 10.1109/SP.2017.49 pp. 39–57. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7958570 [Page 20.]

[36] K. Sethi, E. Sai Rupesh, R. Kumar, P. Bera, and Y. Venu Madhav, "A context-aware robust intrusion detection system: a reinforcement learning-based approach," vol. 19, no. 6, 2020. doi: 10.1007/s10207-019-00482-7. ISSN 1615-5270 pp. 657–678. [Online]. Available: https://doi.org/10.1007/s10207-019-00482-7 [Page 20.]

[37] C. Aggarwal, *Outlier Analysis*. New York N.Y.: Springer, 2013. ISBN 978-1-4614-6395-5. [Online]. Available: https://doi.org/10.1007/978-1-4614-6396-2 [Pages 20, 21, and 22.]

[38] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, 2009. doi: 10.1145/1541880.1541882. [Online]. Available: https://doi.org/10.1145/1541880.1541882 [Pages vii, 21, and 22.]

[39] L. R. Halme and R. K. Bauer, "Aint misbehaving–a taxonomy of anti-intrusion techniques," in *18th National Information Systems Security Conference*, 1995, p. 163. [Online]. Available: https://apps.dtic.mil/sti/pdfs/ADA302547.pdf#page=182 [Page 22.]

[40] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," *Eai Endorsed Transactions on Security and Safety*, vol. 3, no. 9, p. e2, 2016. [Online]. Available: http://eprints.eudl.eu/id/eprint/2057/ [Page 23.]

[41] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, 2018. doi: 10.1109/TETCI.2017.2772792. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8264962?casa_token=owWjh5gFNo4AAAAA:1jppqqQJdOHlbkt_m66c80dF5rJ1OMTg5yIAIwtURjVrz4uATqqCugwi1aqCvZVml_XgmqkB [Page 23.]

[42] F. A. Khan, A. Gumaei, A. Derhab, and A. Hussain, "A novel two-stage deep learning model for efficient network intrusion detection," *IEEE Access*, vol. 7, pp. 30 373–30 385, 2019. doi: 10.1109/ACCESS.2019.2899721. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8643036 [Page 23.]

[43] B. Dong and X. Wang, "Comparison deep learning method to traditional methods using for network intrusion detection," in *2016 8th IEEE International Conference on Communication Software and Networks (ICCSN)*, 2016. doi: 10.1109/ICCSN.2016.7586590 pp. 581–585. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7586590?casa_token=AtDB8v2PToUAAAAA:JA-iF3gJ_TLdxVcWKZmoYk5T2TUaFqhwEliutqrxNoyZkLvTHdmVBaOah1AjC6CbefPtMjkf [Page 23.]

[44] O. Al-Jarrah, A. Siddiqui, M. Elsalamouny, P. Yoo, S. Muhaidat, and K. Kim, "Machine-learning-based feature selection techniques for large-scale network intrusion detection," in *2014 IEEE 34th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2014. doi: 10.1109/ICDCSW.2014.14 pp. 177–181. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6888858?casa_token=CHoYH89uANYAAAAA:ccV0Msq-7sLU-8K6cJram3vDNhnUczqC-RJyin8GvYaYqsHEMGNraVLGwwwA9HjLs4ngc4RO [Page 23.]

[45] S. Otoum, B. Kantarci, and H. T. Mouftah, "On the feasibility of deep learning in sensor network intrusion detection," *IEEE Networking Letters*, vol. 1, no. 2, pp. 68–71, 2019. doi: 10.1109/LNET.2019.2901792. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8653348?casa_token=7wz5UEDVIVAAAAAA:1x8FFdMhH022dMQ2q1yoCjnMgJ6PmUrcl83HjRCCeluR-XyXe47S3AtrSZtNTm8RiXaVFGtz [Page 23.]

[46] S. Raza, L. Wallgren, and T. Voigt, "Svelte: Real-time intrusion detection in the internet of things," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2661–2674, 2013. doi: https://doi.org/10.1016/j.adhoc.2013.04.014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570870513001005 [Page 23.]

[47] K. A. da Costa, J. P. Papa, C. O. Lisboa, R. Munoz, and V. H. C. de Albuquerque, "Internet of things: A survey on machine learning-based

intrusion detection approaches," *Computer Networks*, vol. 151, pp. 147–157, 2019. doi: https://doi.org/10.1016/j.comnet.2019.01.023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128618308739 [Page 23.]

[48] S.-F. Lokman, A. T. Othman, and M.-H. Abu-Bakar, "Intrusion detection system for automotive controller area network (can) bus system: a review," vol. 2019, no. 1, 2019. doi: 10.1186/s13638-019-1484-3. ISSN 1687-1499. [Online]. Available: https://doi.org/10.1186/s13638-019-1484-3 [Page 24.]

[49] O. Y. Al-Jarrah, C. Maple, M. Dianati, D. Oxtoby, and A. Mouzakitis, "Intrusion detection systems for intra-vehicle networks: A review," *IEEE Access*, vol. 7, pp. 21 266–21 289, 2019. doi: 10.1109/ACCESS.2019.2894183. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8642311 [Page 24.]

[50] S. C. Kalkan and O. K. Sahingoz, "In-vehicle intrusion detection system on controller area network with machine learning models," in *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2020. doi: 10.1109/ICCCNT49239.2020.9225442 pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9225442 [Page 24.]

[51] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016. ISBN 978-1-931971-32-4 pp. 911–927. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/cho [Page 24.]

[52] W. Choi, K. Joo, H. J. Jo, M. C. Park, and D. H. Lee, "Voltageids: Low-level communication characteristics for automotive intrusion detection system," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 8, pp. 2114–2129, 2018. doi: 10.1109/TIFS.2018.2812149. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8306904 [Page 24.]

[53] K.-T. Cho and K. G. Shin, "Viden: Attacker identification on in-vehicle networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS

’17. New York, NY, USA: Association for Computing Machinery, 2017. doi: 10.1145/3133956.3134001. ISBN 9781450349468 p. 1109–1123. [Online]. Available: https://doi.org/10.1145/3133956.3134001 [Page 24.]

[54] D. Stabili, M. Marchetti, and M. Colajanni, “Detecting attacks to internal vehicle networks through hamming distance,” in *2017 AEIT International Annual Conference*, 2017. doi: 10.23919/AEIT.2017.8240550 pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8240550?casa_token=-XWhyKLucckAAAAA:LVAZCBMdxqNm79M0VKBYxT07r1cBJ3wGKcHHofWzCF1aC2m7iZfd7OK-ulXU1RYv5YXuxnfv [Pages 25 and 26.]

[55] M.-J. Kang and J.-W. Kang, “Intrusion detection system using deep neural network for in-vehicle network security,” in *PLOS ONE*, 2016. doi: 10.1371/journal.pone.0155781. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0155781 [Page 25.]

[56] A. Taylor, S. Leblanc, and N. Japkowicz, “Anomaly detection in automobile control network data with long short-term memory networks,” in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2016. doi: 10.1109/DSAA.2016.20 pp. 130–139. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7796898?casa_token=VJxIJae2T1wAAAAA:i_bUg4rcLYAaTf3oPqtloYhaiSjrUaiy4i71yDoczUv-JtO-a4Mx02p2smf-nJ3bjYWT__Bd [Pages 26, 40, and 43.]

[57] M. D. Hossain, H. Inoue, H. Ochiai, D. Fall, and Y. Kadobayashi, “Lstm-based intrusion detection system for in-vehicle can bus communications,” *IEEE Access*, vol. 8, pp. 185 489–185 502, 2020. doi: 10.1109/ACCESS.2020.3029307. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9216166 [Page 26.]

[58] S. Longari, D. H. Nova Valcarcel, M. Zago, M. Carminati, and S. Zanero, “Cannolo: An anomaly detection system based on lstm autoencoders for controller area network,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1913–1924, 2021. doi: 10.1109/TNSM.2020.3038991. [Online]. Available:

https://ieeexplore.ieee.org/abstract/document/9262960 [Pages 26 and 93.]

[59] A. Taylor, N. Japkowicz, and S. Leblanc, "Frequency-based anomaly detection for the automotive can bus," in *2015 World Congress on Industrial Control Systems Security (WCICSS)*, 2015. doi: 10.1109/WCICSS.2015.7420322 pp. 45–49. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7420322?casa_token=Q5hTTUzYLmgAAAAA:9kKYs0Pe_j3MOAxH3D9nPlUeYSTL5ay09Dxe4wtqlY5vDanJ44-pwWmttKmLuKySoRJsbqYQ [Page 27.]

[60] E. Seo, H. M. Song, and H. K. Kim, "Gids: Gan based intrusion detection system for in-vehicle network," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, 2018. doi: 10.1109/PST.2018.8514157 pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8514157?casa_token=buuyBgoMA9sAAAAA:x0uE9_1Y7mEhr67-LdUHeOLZObNnvuUQnJ41M1N4TdMAWbmL49qYiz1PrcHsc7YxsOjKmuhJ [Pages 27 and 34.]

[61] R. Islam, R. U. D. Refat, S. M. Yerram, and H. Malik, "Graph-based intrusion detection system for controller area networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 3, pp. 1727–1736, 2022. doi: 10.1109/TITS.2020.3025685. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9210546 [Page 27.]

[62] S. Tariq, S. Lee, and S. S. Woo, *CANTransfer: Transfer Learning Based Intrusion Detection on a Controller Area Network Using Convolutional LSTM Network*. New York, NY, USA: Association for Computing Machinery, 2020, p. 1048–1055. ISBN 9781450368667. [Online]. Available: https://doi.org/10.1145/3341105.3373868 [Page 28.]

[63] A. R. Javed, S. u. Rehman, M. U. Khan, M. Alazab, and T. R. G, "Canintelliids: Detecting in-vehicle intrusion attacks on a controller area network using cnn and attention-based gru," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1456–1466, 2021. doi: 10.1109/TNSE.2021.3059881. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9359538 [Pages 28 and 93.]

[64] M. Hanselmann, T. Strauss, K. Dormann, and H. Ulmer, “Canet: An unsupervised intrusion detection system for high dimensional can bus data,” *IEEE Access*, vol. 8, pp. 58 194–58 205, 2020. doi: 10.1109/ACCESS.2020.2982544. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9044377 [Page 28.]

[65] R. Bhatia, V. Kumar, K. Serag, Z. B. Celik, M. Payer, and D. Xu, “Evading voltage-based intrusion detection on automotive can,” in *Network and Distributed System Security Symposium (NDSS)*, 2021. doi: 10.14722/ndss.2021.23013. [Online]. Available: https://beerkay.github.io/papers/Berkay2021DuetNDSS.pdf [Pages 29 and 30.]

[66] S. U. Sagong, X. Ying, A. Clark, L. Bushnell, and R. Poovendran, “Cloaking the clock: Emulating clock skew in controller area networks,” in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018. doi: 10.1109/ICCPS.2018.00012 pp. 32–42. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8443719 [Page 30.]

[67] M. A. Ayub, W. A. Johnson, D. A. Talbert, and A. Siraj, “Model evasion attack on intrusion detection systems using adversarial machine learning,” in *2020 54th Annual Conference on Information Sciences and Systems (CISS)*, 2020. doi: 10.1109/CISS48834.2020.1570617116 pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9086268 [Page 30.]

[68] Y. Li, J. Lin, and K. Xiong, “An adversarial attack defending system for securing in-vehicle networks,” in *2021 IEEE 18th Annual Consumer Communications Networking Conference (CCNC)*, 2021. doi: 10.1109/CCNC49032.2021.9369569 pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9369569 [Pages 30 and 31.]

[69] MIT Lincoln Lab, “Kdd cup 1999: Computer network intrusion detection,” 1999. [Online]. Available: https://www.kdd.org/kdd-cup/view/kdd-cup-1999/Data [Page 34.]

[70] M. E. Verma, M. D. Iannacone, R. A. Bridges, S. C. Hollifield, B. Kay, and F. L. Combs, “ROAD: the real ORNL automotive dynamometer controller area network intrusion detection dataset (with

a comprehensive CAN IDS dataset survey & guide)," *CoRR*, vol. abs/2012.14600, 2020. [Online]. Available: https://arxiv.org/abs/2012.14600 [Page 34.]

[71] G. Dupont, J. Den Hartog, S. Etalle, and A. Lekidis, "Evaluation framework for network intrusion detection systems for in-vehicle can," in *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*, 2019. doi: 10.1109/ICCVE45908.2019.8965028 pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8965028 [Page 34.]

[72] H. Lee, S. H. Jeong, and H. K. Kim, "Otids: A novel intrusion detection system for in-vehicle network by using remote frame," in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, 2017. doi: 10.1109/PST.2017.00017 pp. 57–5709. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8476919 [Page 34.]

[73] M. L. Han, B. I. Kwak, and H. K. Kim, "Anomaly intrusion detection method for vehicular networks based on survival analysis," *Vehicular Communications*, vol. 14, pp. 52–63, 2018. doi: https://doi.org/10.1016/j.vehcom.2018.09.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214209618301189 [Page 34.]

[74] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002. doi: https://doi.org/10.1613/jair.953. [Online]. Available: https://www.jair.org/index.php/jair/article/view/10302 [Page 36.]

[75] W. Jiang, H. Li, S. Liu, X. Luo, and R. Lu, "Poisoning and evasion attacks against deep learning algorithms in autonomous vehicles," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 4, pp. 4439–4449, 2020. doi: 10.1109/TVT.2020.2977378. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9019666 [Pages 37 and 38.]

[76] G. Apruzzese, M. Andreolini, L. Ferretti, M. Marchetti, and M. Colajanni, "Modeling realistic adversarial attacks against network intrusion detection systems," *CoRR*, vol. abs/2106.09380, 2021. [Online]. Available: https://arxiv.org/abs/2106.09380 [Pages 37 and 38.]

[77] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/viewPaper/14806 [Page 39.]

[78] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html [Page 40.]

[79] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html [Page 40.]

[80] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Artificial intelligence safety and security*. Chapman and Hall/CRC, 2018, pp. 99–112. [Online]. Available: https://www.taylorfrancis.com/chapters/edit/10.1201/9781351251389-8/adversarial-examples-physical-world-alexey-kurakin-ian-goodfellow-samy-bengio [Page 49.]

Appendix A

Code

A.1 Fast Gradient Sign Method

Algorithm 5 Fast Gradient Sign Method Python

```
Require: model_fn, x, eps, norm, features, loss_fn, clip_min, clip_max, y
# Compute the gradient of the loss_fn on model_fn w.r.t. x
grad = compute_gradient(model_fn, loss_fn, x, y)

# Limit gradient to chosen features
limit_grad = features * grad

# Test additional limitation for features that have hit
# max or min value and will be updated again
# This stops infinitely updating a single value
# (especially when using 1-norm)
limitations = np.ones(len(features))
for i=0 in range(len(limit_grad)) do
    if (x[i] >= clip_max[i]) and (limit_grad[i] > 0) then
        limitations[i] = 0
    else if (x[i] <= clip_min) and (limit_grad[i] < 0) then
        limitations[i] = 0

limit_grad = limit_grad * limitations

# Compute perturbation based on gradient, epsilon and the desired norm
optimal_perturbation = optimize_linear(limit_grad, eps, norm)

# Add perturbation to original sample
adv_x = x + optimal_perturbation

# Clip values of adversarial sample to max and min values
adv_x = clip_by_value(adv_x, clip_min, clip_max)
return adv_x
```

Algorithm 6 FGSM Loop Python

```
Require: model_fn, x, eps, norm, features, loss_fn, clip_min, clip_max, y, max_iter
# Setup necessary variables
iteration = 0
adv_x = x
x_label = model_fn.predict(adv_x)

# Set placeholder for rounded version of adversarial sample
x_round = adv_x
x_round_label = x_label

# Loop till either adversarial sample is successful or
# maximum number of iterations is reached
while x_round_label != y and iteration < max_iter do
    #compute adversarial sample
    adv_x = FGSM(model_fn, adv_x, eps, norm, features, loss_fn, clip_min, clip_max, y)

    # compute new label
    x_label = model_fn.predict(adv_x)

    # if sample misclassifies, round values and check again
    if label of adv_x == y then
        # round all values
        x_round = round(adv_x)
        # reset dTIME to non-rounded value
        x_round[0][12] = adv_x[0][12]
        x_round_label = model_fn.predict(x_round)

    # Update iteration counter
    iteration = iteration + 1;

# Add check if maximum iterations reached
if iteration == max_iter then
    # update to latest sample
    x_round = round(adv_x)
return x_round, iteration
```

A.2 SOTA-CNN Frame Builder

Algorithm 7 Frame Builder for the SOTA-CNN model data Python (Part 1)

```
Require: $X, Y, nrAttacks$
length = len(X)-28

# setup required variables
X_frames = np.zeros((length, 29, 29))
Y_frames = np.zeros(length)

# Setup Initial Frame
initial_frame = np.zeros((29, 29))
attack_counter = 0.0
initial_label = 0.0
for i=0 in range(29) do:
    initial_frame[i][0:11] = X[i][0:11]

    #add label values
    attack_counter = attack_counter + Y[i]

if attack_counter > nrAttacks then
    initial_label = 1.0

X_frames[0][:][:] = initial_frame
Y_frames[0] = initial_label
```

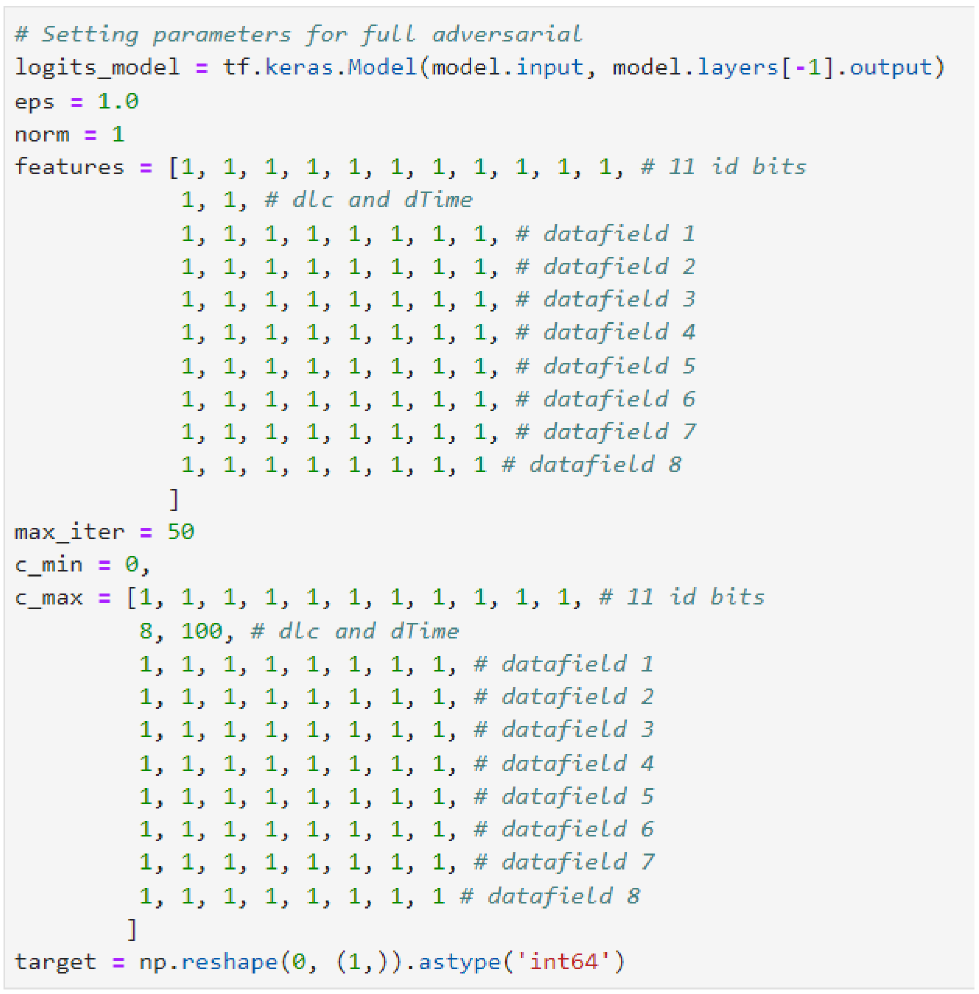
Algorithm 8 Frame Builder for the SOTA-CNN model data Python (Part 2)

```
Require: $X, Y, nrAttacks$
# compute other frames based on initial frame
temp_frame = np.copy(initial_frame)
for i in range(29, len(X)) do
    temp_label = 0.0
    # remove first row and adjust attack counter accordingly
    attack_counter = attack_counter - Y[i-28]
    temp_frame = np.delete(temp_frame , 0, axis=0)

    # add new row and adjust attack counter
    attack_counter = attack_counter + Y[i]
    new_row = np.zeros(29)
    new_row[0:11] = X[i][0:11]
    temp_frame = np.append(temp_frame, [new_row], axis = 0)

    #check if attacks in frame succeeds threshold
    if attack_counter > nrAttacks then
        temp_label = 1.0

    X_frames[i-28][:][:] = temp_frame
    Y_frames[i-28] = temp_label
return X_frames, Y_frames
```

Appendix B

Dataset Parameters

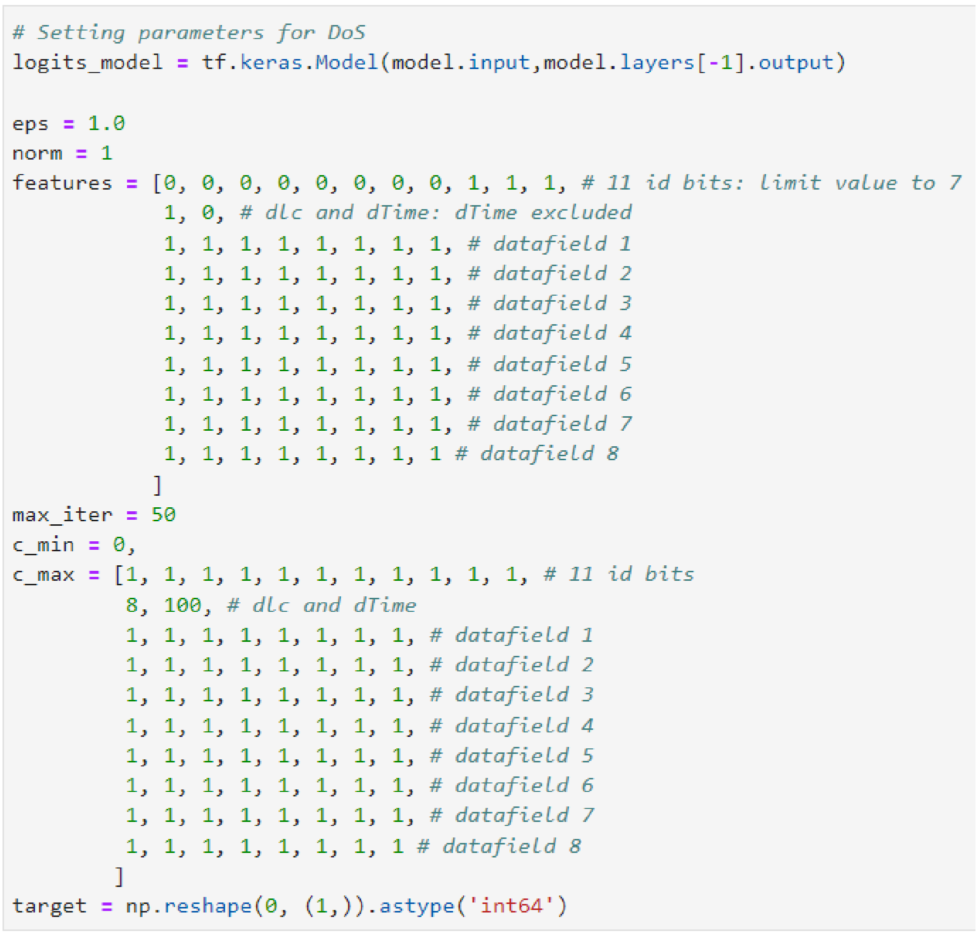
B.1 Full dataset



```
# Setting parameters for full adversarial
logits_model = tf.keras.Model(model.input, model.layers[-1].output)
eps = 1.0
norm = 1
features = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, # 11 id bits
            1, 1, # dlc and dTime
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 1
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 2
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 3
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 4
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 5
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 6
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 7
            1, 1, 1, 1, 1, 1, 1, 1 # datafield 8
           ]
max_iter = 50
c_min = 0,
c_max = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, # 11 id bits
         8, 100, # dlc and dTime
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 1
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 2
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 3
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 4
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 5
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 6
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 7
         1, 1, 1, 1, 1, 1, 1, 1 # datafield 8
        ]
target = np.reshape(0, (1,)).astype('int64')
```

Figure B.1: All parameters for FGSM for Full Dataset

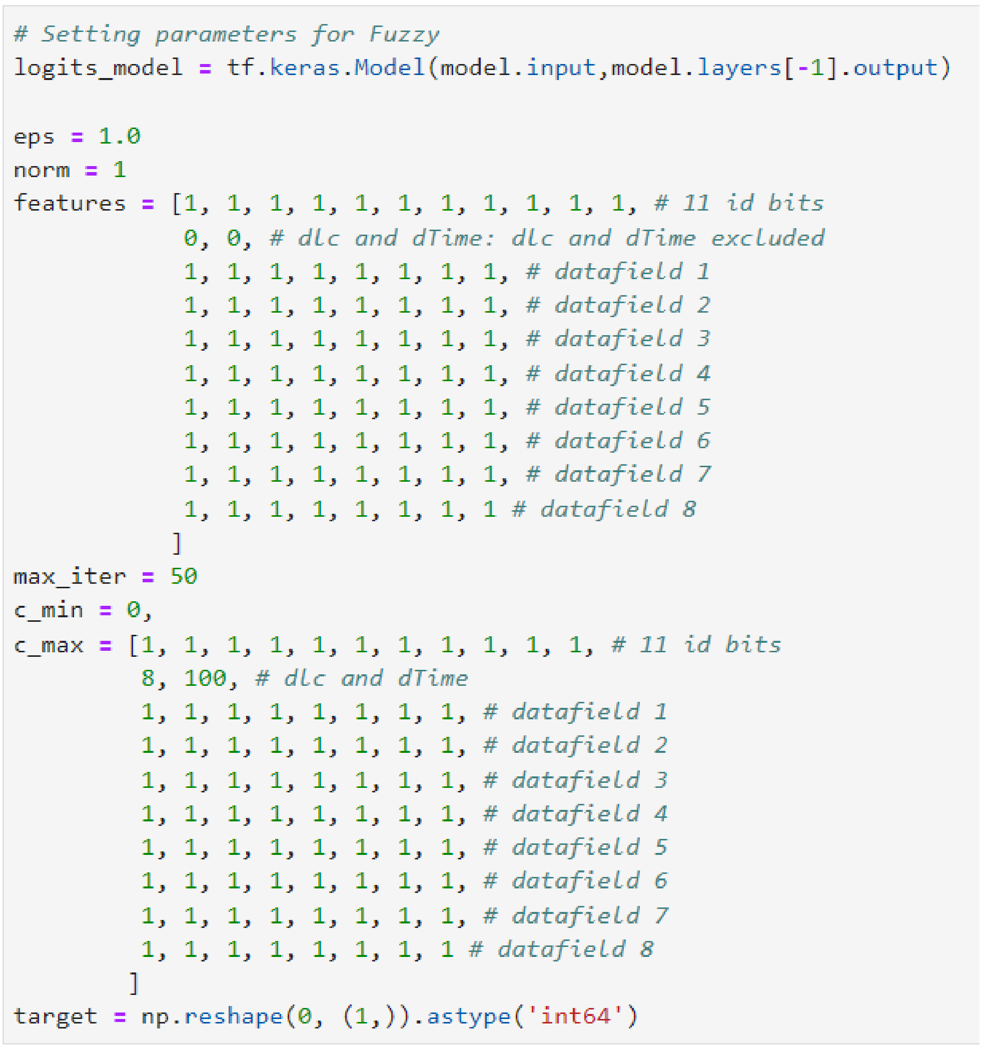
B.2 DoS dataset



```
# Setting parameters for DoS
logits_model = tf.keras.Model(model.input,model.layers[-1].output)

eps = 1.0
norm = 1
features = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, # 11 id bits: limit value to 7
            1, 0, # dlc and dTime: dTime excluded
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 1
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 2
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 3
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 4
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 5
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 6
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 7
            1, 1, 1, 1, 1, 1, 1, 1 # datafield 8
           ]
max_iter = 50
c_min = 0,
c_max = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, # 11 id bits
         8, 100, # dlc and dTime
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 1
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 2
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 3
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 4
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 5
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 6
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 7
         1, 1, 1, 1, 1, 1, 1, 1 # datafield 8
        ]
target = np.reshape(0, (1,)).astype('int64')
```

Figure B.2: All parameters for FGSM for DoS Dataset

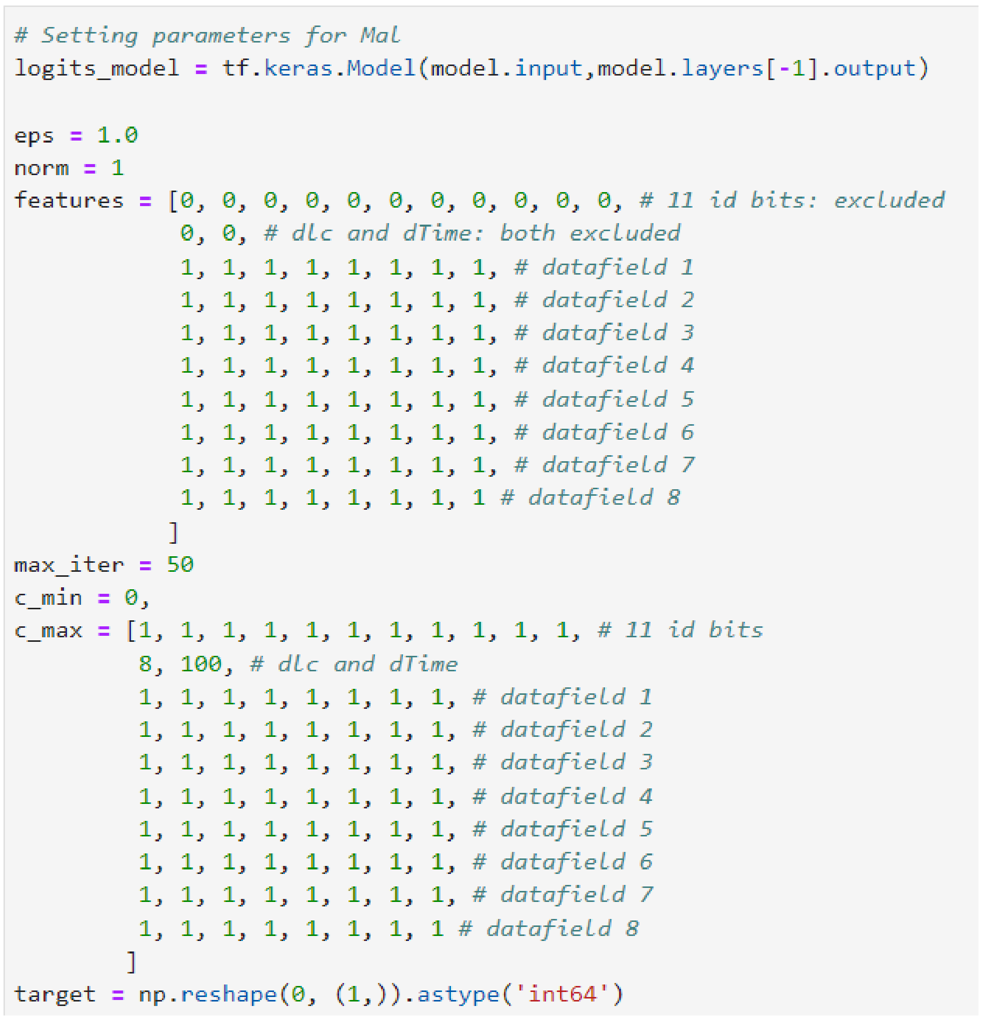
B.3 Fuzzy dataset



```
# Setting parameters for Fuzzy
logits_model = tf.keras.Model(model.input,model.layers[-1].output)

eps = 1.0
norm = 1
features = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, # 11 id bits
            0, 0, # dlc and dTime: dlc and dTime excluded
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 1
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 2
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 3
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 4
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 5
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 6
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 7
            1, 1, 1, 1, 1, 1, 1, 1 # datafield 8
           ]
max_iter = 50
c_min = 0,
c_max = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, # 11 id bits
         8, 100, # dlc and dTime
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 1
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 2
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 3
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 4
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 5
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 6
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 7
         1, 1, 1, 1, 1, 1, 1, 1 # datafield 8
        ]
target = np.reshape(0, (1,)).astype('int64')
```

Figure B.3: All parameters for FGSM for Fuzzy Dataset

B.4 Malfunction dataset



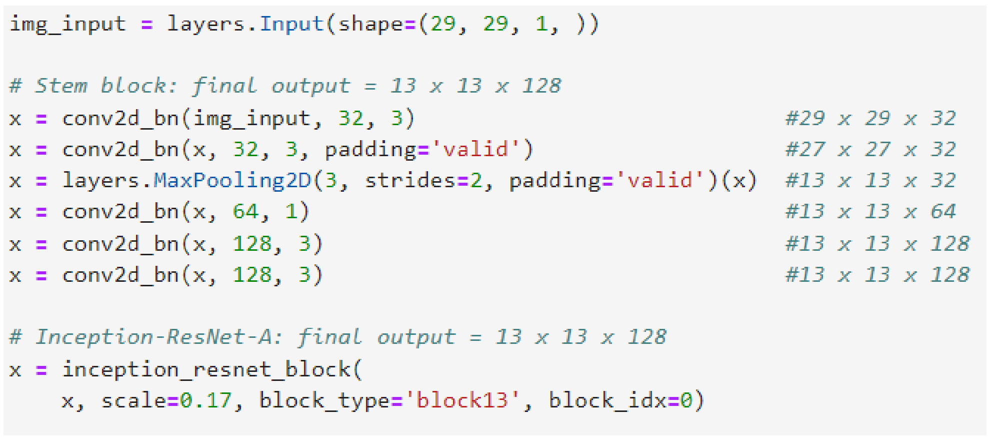
```
# Setting parameters for Mal
logits_model = tf.keras.Model(model.input,model.layers[-1].output)

eps = 1.0
norm = 1
features = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, # 11 id bits: excluded
            0, 0, # dlc and dTime: both excluded
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 1
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 2
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 3
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 4
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 5
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 6
            1, 1, 1, 1, 1, 1, 1, 1, # datafield 7
            1, 1, 1, 1, 1, 1, 1, 1 # datafield 8
           ]
max_iter = 50
c_min = 0,
c_max = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, # 11 id bits
         8, 100, # dlc and dTime
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 1
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 2
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 3
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 4
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 5
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 6
         1, 1, 1, 1, 1, 1, 1, 1, # datafield 7
         1, 1, 1, 1, 1, 1, 1, 1 # datafield 8
        ]
target = np.reshape(0, (1,)).astype('int64')
```

Figure B.4: All parameters for FGSM for Malfunction Dataset

Appendix C

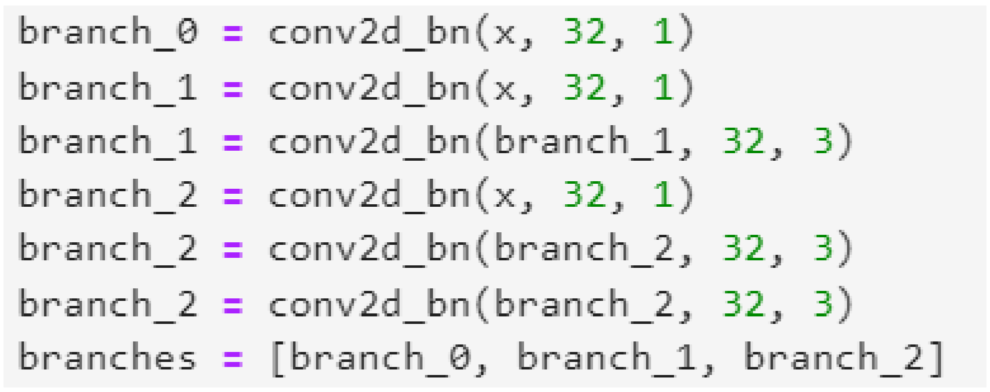
CNN Implementation

In the figures below, parts of the python implementation of the SOTA-CNN model can be seen. The parts include the Stem, Inception-ResNet-A, Inception-ResNet-B, Reduction-A, Reduction-B and the final layers. Note that the implementation for the Inception blocks is split up, where Figure C.2 and Figure C.3 show the branches of respectively the Inception-ResNet-A and Inception-ResNet-B blocks. Figure C.4 shows the implementation of the concatenation layers of both blocks.



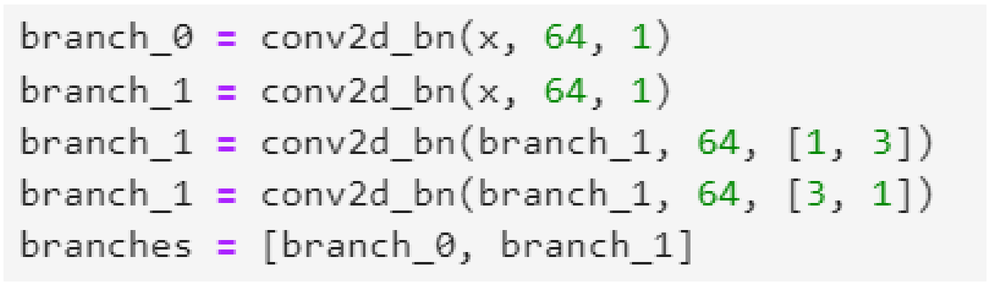
```
img_input = layers.Input(shape=(29, 29, 1, ))

# Stem block: final output = 13 x 13 x 128
x = conv2d_bn(img_input, 32, 3)                           #29 x 29 x 32
x = conv2d_bn(x, 32, 3, padding='valid')                  #27 x 27 x 32
x = layers.MaxPooling2D(3, strides=2, padding='valid')(x)  #13 x 13 x 32
x = conv2d_bn(x, 64, 1)                                   #13 x 13 x 64
x = conv2d_bn(x, 128, 3)                                  #13 x 13 x 128
x = conv2d_bn(x, 128, 3)                                  #13 x 13 x 128

# Inception-ResNet-A: final output = 13 x 13 x 128
x = inception_resnet_block(
    x, scale=0.17, block_type='block13', block_idx=0)
```

Figure C.1: SOTA-CNN Stem implementation



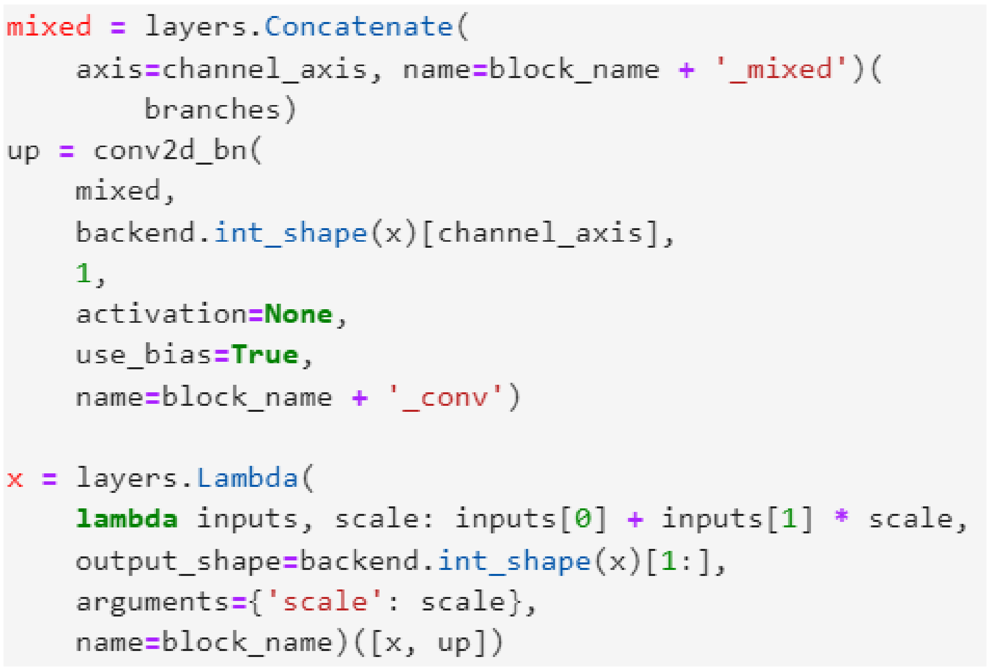
```
branch_0 = conv2d_bn(x, 32, 1)
branch_1 = conv2d_bn(x, 32, 1)
branch_1 = conv2d_bn(branch_1, 32, 3)
branch_2 = conv2d_bn(x, 32, 1)
branch_2 = conv2d_bn(branch_2, 32, 3)
branch_2 = conv2d_bn(branch_2, 32, 3)
branches = [branch_0, branch_1, branch_2]
```

Figure C.2: SOTA-CNN Inception-ResNet-A implementation



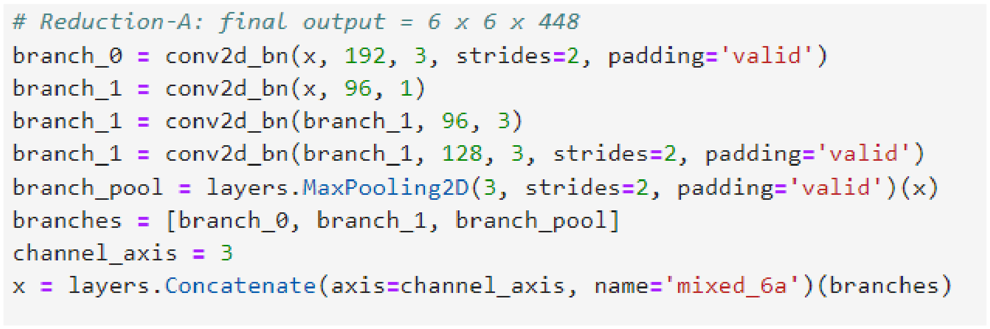
```
branch_0 = conv2d_bn(x, 64, 1)
branch_1 = conv2d_bn(x, 64, 1)
branch_1 = conv2d_bn(branch_1, 64, [1, 3])
branch_1 = conv2d_bn(branch_1, 64, [3, 1])
branches = [branch_0, branch_1]
```

Figure C.3: SOTA-CNN Inception-ResNet-B implementation



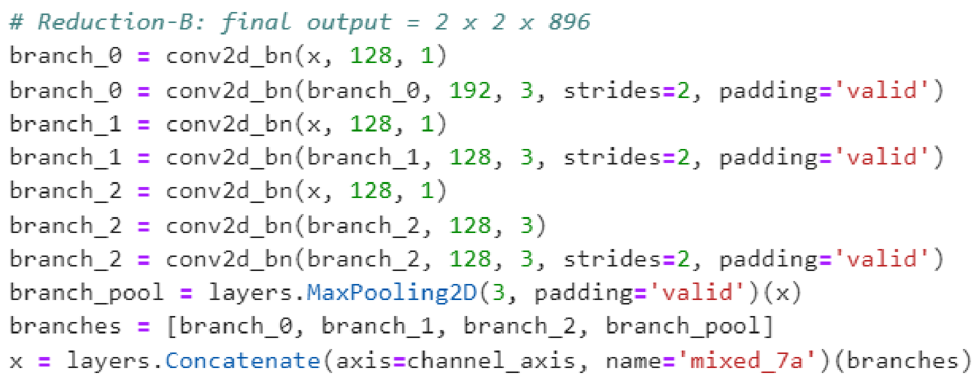
```
mixed = layers.Concatenate(
    axis=channel_axis, name=block_name + '_mixed')(
        branches)
up = conv2d_bn(
    mixed,
    backend.int_shape(x)[channel_axis],
    1,
    activation=None,
    use_bias=True,
    name=block_name + '_conv')

x = layers.Lambda(
    lambda inputs, scale: inputs[0] + inputs[1] * scale,
    output_shape=backend.int_shape(x)[1:],
    arguments={'scale': scale},
    name=block_name)([x, up])
```

Figure C.4: SOTA-CNN Inception-ResNet concat layers implementation



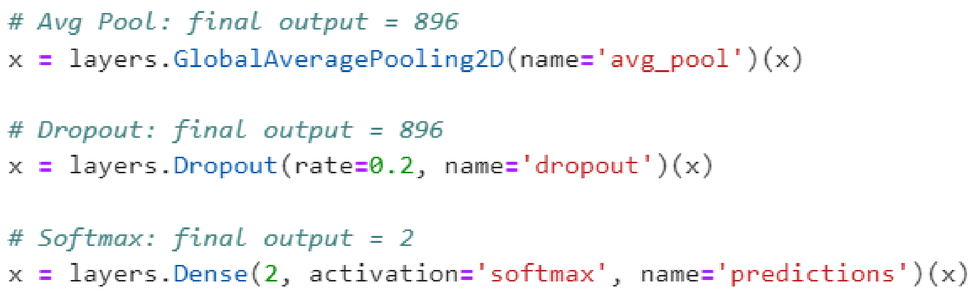
```
# Reduction-A: final output = 6 x 6 x 448
branch_0 = conv2d_bn(x, 192, 3, strides=2, padding='valid')
branch_1 = conv2d_bn(x, 96, 1)
branch_1 = conv2d_bn(branch_1, 96, 3)
branch_1 = conv2d_bn(branch_1, 128, 3, strides=2, padding='valid')
branch_pool = layers.MaxPooling2D(3, strides=2, padding='valid')(x)
branches = [branch_0, branch_1, branch_pool]
channel_axis = 3
x = layers.Concatenate(axis=channel_axis, name='mixed_6a')(branches)
```

Figure C.5: SOTA-CNN Reduction-A implementation



```
# Reduction-B: final output = 2 x 2 x 896
branch_0 = conv2d_bn(x, 128, 1)
branch_0 = conv2d_bn(branch_0, 192, 3, strides=2, padding='valid')
branch_1 = conv2d_bn(x, 128, 1)
branch_1 = conv2d_bn(branch_1, 128, 3, strides=2, padding='valid')
branch_2 = conv2d_bn(x, 128, 1)
branch_2 = conv2d_bn(branch_2, 128, 3)
branch_2 = conv2d_bn(branch_2, 128, 3, strides=2, padding='valid')
branch_pool = layers.MaxPooling2D(3, padding='valid')(x)
branches = [branch_0, branch_1, branch_2, branch_pool]
x = layers.Concatenate(axis=channel_axis, name='mixed_7a')(branches)
```

Figure C.6: SOTA-CNN Reduction-B implementation



```
# Avg Pool: final output = 896
x = layers.GlobalAveragePooling2D(name='avg_pool')(x)

# Dropout: final output = 896
x = layers.Dropout(rate=0.2, name='dropout')(x)

# Softmax: final output = 2
x = layers.Dense(2, activation='softmax', name='predictions')(x)
```

Figure C.7: SOTA-CNN final layers implementation