Doctoral Dissertation

In-Vehicle Network Attack Detection Using Deep Neural Networks Trained On Features Extracted From CAN Data

Araya Kibrom Desta

August 28, 2022

Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Araya Kibrom Desta

Thesis Committee:

Professor Kazutoshi Fujikawa	(Supervisor)
Professor Youki Kadobayashi	(Co-supervisor)
Professor Yuichi Hayashi	(Co-supervisor)
Associate Professor Ismail Arai	(Co-supervisor)

In-Vehicle Network Attack Detection Using Deep Neural Networks Trained On Features Extracted From CAN Data*

Araya Kibrom Desta

Abstract

Advanced Electronic Control Units (ECU) have been included in automobiles in recent years to ensure safe and comfortable driving. Unlike in the past, when ECUs were connected by point-to-point wire, today's automobiles employ a *de facto* networking standard known as the controller area network (CAN). CAN is vulnerable to cyber attacks because, despite its capacity to identify errors, maintain data integrity, and maintain consistency, it fails to secure the network by utilizing authentication, encryption, and network segmentation. The dissertation proposes an intrusion detection systems (IDS) for the CAN bus using deep learning that is trained on the CAN bus data. Four methods are experimented to secure the CAN bus. In the first two methods, the arbitration ID of the CAN frames is used to train Long Short-Term Memory Networks (LSTM) and Convolutional Neural Networks (CNN). The LSTM-based IDS is trained to learn the sequence of arbitration IDs in the CAN bus. The trained model is used to predict the future sequence of arbitration IDs with wrong predictions being flagged as an attack. In such a way, LSTM-based IDS has improved the conventional IDS method that studies arbitration ID patterns. Even though LSTM managed to improve the conventional method performance in detecting spoofing the gear attack and spoofing the RPM attacks, its results are not very accurate. The spoofing attack detection in the gear has improved from F1 score of 0.59 to 0.953 and spoofing

*Doctoral Dissertation, Graduate School of Information Science,
Nara Institute of Science and Technology, August 28, 2022.

attack detection in the RPM has improved from F1 score of 0.628 to 0.63. CNN-based IDS called Rec-CNN is proposed as an improvement to the LSTM-based IDS. Images generated using recurrence plots from the CAN bus arbitration IDs are used to train the CNN architecture. Using recurrence plots helps in capturing the temporal data in the CAN bus data through images. Using these images of recurrence plots, the experiment is done on how CNNs can easily be trained to classify attack and benign sequences of arbitration ID for a secure CAN bus communication. Both the works use CAN arbitration IDs to train LSTMs and CNNs. If the arbitration ID is not affected during an attack, attacks will be left undetected. To improve this drawback, two other methods are proposed using the data section of the CAN frame. The first work, named MLIDS, trains an LSTM architecture that is capable of handling the high dimensional CAN bus data without requiring reveres-engineering of the CAN bus data. Training LSTM can be difficult in the CAN bus data as it contains millions of parameters. Our last work called U-CAN is proposed as an improvement to the MLIDS. U-CAN is trained using the hamming distance (HAMD) distribution of CAN frame bits. All the works have been tested against different sets of attacks including fuzzy attacks, drop attacks, denial of service (DoS) attacks, insertion attacks, and spoofing attacks.

Keywords:

Convolutional Neural Networks, Long Short-Term Memory Networks,In-vehcile Networks, Intrusion Detection, IoT Security, Recurrence plots

Contents

List of Figures

1 Introduction

Vehicles have undergone a succession of developments since the advent of the first steam-powered automobile to ensure safe and enjoyable driving. Electronic control units (ECU) in modern vehicles enable them to do remarkable tasks, such as driver assistance technology, making them smarter. Ten years ago only luxurious cars had these advanced ECUs that can reach up to 100. But in recent years the number of ECUs even in average cars surpasses 100 ECUs while those of luxurious cars can go beyond 150 [1]. These ECUs can react to a user's engagement with the vehicle or they can operate on their own. Unlike in the past, when networks of ECUs were built via point-to-point wiring, most automobiles today employ networking and the communication between the ECUs is governed by a networking protocol. There are quite a few in-vehicle network protocols used by modern vehicles, the networking standard a car uses is decided by the manufacturers according to their needs. One such protocol that is most widely used is the controller area network (CAN). CAN serial bus system was introduced by Robert Bosch GmbH at the Society of Automotive Engineers (SAE) congress [2]. CAN define a standard for efficient and reliable communication between sensor, actuator, controller, and other nodes in real-time applications. CAN is a de facto standard in a variety of networked embedded control systems. CAN protocol standardizes the physical and data link layers [3]. Due to its efficiency in carrying out diagnostics and coordination of operations in separate subsystems, it easily proliferated not only in in-vehicle networks but also in medical apparatuses, agriculture, etc [4].

ECUs can also communicate with devices outside the vehicle for various information exchanges for the purposes of awareness-based driving. This information exchange between the vehicles and the outside world can be a target of attack [5]. In spite of CAN's design that is lightweight, robust, fast capable of running in low-performance electronic devices, its current implementation supports no au-

thentication or encryption to enhance the integrity of messages sent from different ECUs [6]. This shows us the need for a system that can monitor the CAN network traffic for any kind of suspicious activity. Such systems are called intrusion detection systems (IDS), which we are mainly focusing on in this thesis.

1.1 The need for an in-vehicle network intrusion detection system

Drivers nowadays can get road information through Vehicular Ad-hoc NETworks (VANETs). This types of networks include Vehicle-to-Vehicle (V2V) and Vehicle-Infrastructure (V2I) networks that are used to assist drivers by providing information about other vehicles and the surrounding environment [7]. Aside from these networks, vehicule manufacturers might also provide services such as Global Positioning Systems (GPS) and related information through cellular networks that are integrated with the vehicles. These types of networks can be used as a means to get into the in-vehicle networks. Although the security of these networks is outside the subject of this study, regardless the source of the attack, an intrusion detection system (IDS) should be able to identify these attacks aimed at the in-vehicle networks.

Error detection, data integrity, and data consistency are implemented in the CAN, but it does not apply security measures to secure the network communication. CAN lucks authentication, data encryption, and network segmentation [8, 9]. Researchers have considered this vulnerability when remotely controlling cars, including a 2014 Jeep Cherokee [10], which showed the possibility of remote attacks against unaltered vehicles. They have managed to send attack packets through the compromised head unit (radio) to the CAN bus because the head unit in the 2014 Jeep Cherokee car is connected to the CAN bus. Similar research from the Keen Security Lab also showed how a Tesla Model S/X could be remotely attacked [9]. Moreover, CAN uses a broadcast communication protocol, that enables attackers to easily snoop on all communication channels or send packets to any other node. It uses arbitration to control the priority of resource usage in the arriving packets. This makes it vulnerable to DoS attacks that can be performed by continuously flooding the bus with high-priority packets [11, 12]. It is also not possible to know

the source of certain CAN frames. These and other security vulnerabilities have been studied by researchers which lead to us to believe there is a need for an intrusion detection system in the CAN bus [13, 14].

1.2 Research contributions

The dissertation mainly deals with training deep neural networks using CAN bus data for intrusion detection. CAN has an arbitration ID part and a data part that can be used as a training data. The methodologies in this work are grouped in to CAN arbitration ID based and CAN data based intrusion detection. In the first work, we have improved the security drawbacks of the CAN bus by proposing an intrusion detection system (IDS) that is trained using the sequence of CAN packets' arbitration IDs. In addition to other information, CAN packets have an arbitration ID that is used to control the priority of CAN packets. Our intrusion detection method extracts the IDs to train Long Short-term Memory Networks (LSTM). Once, the LSTM network learns the pattern of the IDs, if there is any, it is used to predict an arbitration ID that might appear after a certain sequence of arbitration IDs. Relying on the predicted arbitration ID, an anomaly signal is prepared from a softmax probability of all the available classes (all the arbitration IDs). An anomaly is detected using two ways. The first approach compares the probability values of all the classes and selects the one with the highest probability as a predicted arbitration ID. The predicted ID is then compared with the true ID for anomaly detection. The second approach gets an aggregated log loss value of the predicted classes that will be later compared with a predefined anomaly signal threshold to detect for intrusions. The conventional method proposed by [15] trains a single transition matrix that will be used to test the possible transitions between two different IDs. Even if this performs with near perfect precision value (0.999), it has a very low recall value (0.4) as it is impossible to grab millions of arbitration ID sequences in a single transition matrix. In this work, we have proposed an IDS using LSTM that greatly outperforms the conventional method. The main contribution of this research study is to improve the anomaly detection accuracy of the conventional method. In addition, we tested both the conventional and trained network against a publicly available dataset. The method is referenced

as LSTM-IDS in rest of the dissertation.

The IDS in the CAN bus must be fast and effective, capable of identifying any kind of attacks targeting the in-vehicle network. The LSTM-based system was incapable of learning the sequence of arbitration IDs with a perfect accuracy. In our second work, we took advantage of advancements in deep neural networks for intrusion detection. Deep learning has shown promising results in various fields, especially in image detection and classifications most notably, CNNs [16]. In this work, we show how CNNs performance in images could solve intrusion detection problems. To do so, we need to change the intrusion detection problem to an image classification problem. Previous research has employed CNNs for in-vehicle network intrusion detection [17]. It builds a 2D grid data frame from arbitration IDs as an input to the deep CNN network. The generated images are then trained using a simplified Inception-ResNet model for in-vehicle network intrusion detection. However, the images generated in this method do not grasp the temporal dependency in the sequence of arbitration IDs in the CAN bus. As far as our knowledge, this is the only research that uses CNNs trained on arbitration IDs for in-vehicle network security. Improving the Inception-ResNet-based method would require an image generation-algorithm that can also keep the temporal dependency. We have solved this issue using recurrence plots to create an image from the CAN bus's time-series data. First packets that have arrived in the CAN bus are collected. These packets contain signal information and arbitration IDs. The arbitration IDs are used to control the priory of using CAN bus resources with lower arbitration IDs having a high priority. To generate the images, extraction of these sequence of arbitration IDs is required. With the sequence of arbitration IDs, a square matrix is created using a recurrence plot algorithm, but unlike normal recurrence plots [18], categorical recurrence plots are used. The main reason for this is for plots to show no special effect to closer encoding of arbitration IDs than the further encoding of arbitration IDs. The formal operation of recurrence plots is performed if and only if the two arbitration IDs are encoded similarly. Using these images of recurrence plots, experiment is done on how CNNs can easily be trained to classify attack and benign sequences of arbitration ID for a secure CAN bus communication.

Table 1.1 also shows the contribution of our works that study the sequential

Table 1.1: Methods trained on the CAN arbitration IDs

Method	**Matrix-transition-based [15]**	**LSTM-IDS (Our work)**	**GAN-based [19]**	**ResNet-based [17]**	**Rec-CNN (our work)**
Features used	Arbitration ID	Arbitration ID	Arbitration ID	Arbitration ID	Arbitration ID
Method	Transition matrix	LSTM	GANs	ResNets	CNN
Attacks tested	Spoofing gear spoofing RPM	Spoofing gear Spoofing RPM	Spoofing gear Spoofing RPM	Drop Fuzzy Insertion Spoofing RPM Spoofing gear DoS	Drop Fuzzy Insertion Spoofing RPM Spoofing gear DoS
Using the method	Low performance Fast	Improved performance Not accurate	Improved Performance Lower accuracy than ResNet-based	Better Performance Temporal information not used	Improved accuracy of all Recurrence plots used to incorporate temporal information
Attack detection	Worst	Improved Matrix-transition-based	Improved LSTM-IDS	Improved GAN-based	Best

information of arbitration IDs in CAN frames. The subsequent work is added as an improvement to the proceeding work. In the table, "Features used" is arbitration ID for all the methodologies. "Attacks tested" refers to the different types of attacks tested. The performance of the existing methods in comparison with our work is shown in the "Using method" row reflecting Rec-CNN outperforming all our previous works and existing research. A detailed comparison of our work with the existing research is explained in detail in the related works section.

The CAN arbitration IDs are used in the preceding two works, however if the attacker can retain the sequence of arbitration IDs while changing the data section of the CAN, the IDS will be bypassed. To overcome the issue, we have proposed an intrusion detection system using LSTM using the data part of the CAN data. Unlike the conventional methods which either require reverse-engineering the CAN bus data or multiple architecture implementation, the work requires no reverse-engineering and only has a single anomaly signal for a window of detection. It is tested on insertion, drop and fuzzy attacks to analyse the security aspects of the CAN bus. The proposed method effectively identified insertion and drop attacks with 100% precision except for some drop attacks with a very short detection windows. Drop attacks can easily be detected with 100% precision when the detection windows is above 1 second, but when the detection window is below 1 second its precision drops low as most of the packets in that short time window might be dropped.

Because the CAN data contains so many features, training the LSTM-based

Table 1.2: Methods trained on the data part of CAN frames

Method	**CANET [20]**	**MLIDS (Our work)**	**U-CAN Our work**
Features used	CAN data	CAN data	CAN data
Method	LSTM	LSTM	CNN
Attacks tested	Playback	Spoofing gear	Platuea
	Flooding	Spoofing RPM	DoS
	Suppress	Insertion	Fuzzy
	Continuous	Drop	Spoofing RPM
	Plateau attacks	Spoofing gear	
Using the method	Reverese engineering requirement	Works in raw CAN data	Works in both raw and reverse-engineered
	Too big of a model to train	Too big of a model to train and deploy	Lower parameters to train
Attack detection	Low	High	High

IDS can be problematic. If we have 20 arbitration IDs and a data frame size of 64 bits, the total features are 20×64, making it too large a feature to search the LSTM hyperparameters. U-CAN is an improvement to the LSTM-based IDS that is trained on the data-part of the CAN frames. It takes the advancements in CNN's image segmentation to in-vehicle network intrusion detection. Previous research works have used CNNs to detect CAN intrusions, but the vast majority of studies have dealt with either raw CAN data or reverse-engineered CAN data. Unlike previous studies, U-CAN can be used for both types of data. U-CAN uses a hamming distance (HAMD) distribution of CAN frame bits to deal with raw CAN frames that are used to train a model. The trained model can be deployed to listen in the OBD II port of vehicles for intrusion detection. We have also added a rule-based system that checks the HAMD of counter bits in a CAN frame. The rule-based system flags any sequence that deviates from a predefined hamming value. For reverse-engineered CAN frames, we applied a saliency detection algorithm before feeding the data to U-CAN. This helps us to easily segment attack windows of the CAN signal sequences.

Table 1.2 shows the summary of the second group of work that uses the data part of CAN frames. The methods as shown in the table use LSTMs, and CNNs that are trained using features preprocessed from the data part of CAN frames. Various types of attacks have been tested. The detailed description of the methodologies with regard to previous research works is shown in the related works section.

1.3 Organization of this dissertation

The dissertation's remaining chapters are arranged as follows. In chapter 2, the CAN bus is described in depth, along with any security flaws and potential entry points for hackers looking to access in-vehicle networks. This chapter also provides an introduction to related works that are pertinent to our research. The specifics of our initial research, which concerns the sequence of arbitration IDs, are discussed in full in chapter 3. The chapter describes how LSTM may be used to identify CAN bus attacks. Given that LSTMs may be challenging to train, in chapter 4 we looked at how CNNs could be used to train models that could distinguish between attack and non-attack images that are generated using recurrence plots. Only arbitration IDs are used to train a deep learning model in each of the approaches discussed in chapters 3 and 4. Attacks won't be detected if the ID sequence isn't affected. We suggested an LSTM-based IDS in chapter 5 that is trained on the data portion of the CAN frames to address this problem. This technique can manage the CAN bus data's high-dimensional feature. However, as was previously said, LSTM training can be challenging. Thus, we switched the LSTM to a CNN-based IDS in our most recent proposal in chapter 6. In contrast to chapter 4, this approach is trained on the data portion of the CAN frames. We used a spectral residual algorithm to prepare the CAN data for CNN training. The last chapter provides a summary of all the dissertation's research as well as recommendations on how the suggested approaches could be improved in the future.

2 Background and related work

The available in-vehicle network standards are discussed in this chapter, with an emphasis on the CAN bus. The security challenges of automobiles that use the CAN bus are then discussed in depth. Knowing merely the CAN bus's security issues may not be enough to exploit in-vehicle networks. As a result, we've included the CAN attack surfaces in the following sections. After constructing the research challenge, we included a section for related works, which outlines past research contributions by scholars attempting to address the same problem as ours.

2.1 Controller area network

The ECUs found in vehicles react to drivers' interactions with the host car. The ECUs send a packet to the bus protocol that governs the packet transfer in vehicles. The CAN protocol, SAE J1850 protocol, KWP2000 protocol, (local interconnect network) LIN protocol, media-oriented systems transport (MOST protocol), and FlexRay are among the most used protocols in in-vehicle networks [21]. Even though a protocol in a car is determined by the demands of its makers, the CAN is the de facto standard in today's automobiles. As a result, we'll be focusing on the CAN bus for this study.

Figure 2.1 shows the general structure of in-vehicle network architecture. In the figure, a single CAN is acting as a communication medium for the connected ECUs. In cases where a vehicle integrates a few more communication protocols, the gateway acts as an interface between different connection protocols. As depicted in the figure, the architecture only shows a single CAN network but a vehicle can also have more than one CAN that is connected through a bridge. Despite the number of CAN buses used in a vehicle, most of the packets generated in the

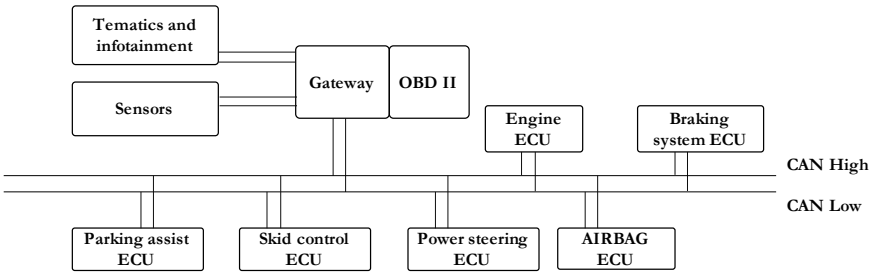


Figure 2.1: In-vehicle network system architecture

SOF	Arbitration	Control	Data	CRC	ACK	EOF
1 bit	11 or 29 bits	6 bits	0-8 bytes	16 bits	2 bits	10 bits

Figure 2.2: CAN frame

ECUs are accessed via the OBD II port that is usually found under the driving wheel of most modern cars [22, 23, 24, 25].

The CAN protocol is a serial communication protocol that was created with real-time communication in mind. Since its first release in the 1980s by BOSCH [26], it got proliferated in vehicles due to its design simplicity and cost-effectiveness. CAN is a multi-master broadcast communication protocol based on arbitration. The arbitration allows highly critical ECUs to be given priority in using the CAN bus resources. The bit size of the arbitration ID is 11 bits in the standard CAN and 29 bits in the extended CAN. The general structure of the CAN frame is shown in Figure. 2.2. It begins with the Start of Frame (SOF) field that is only represented by a single dominant bit. Then the arbitration ID comes with 11 or 29 bits which ECUs use to identify the meaning of the frame to decide on whether to act upon it or not. The control field is a placeholder for the data length code (DLC). The value of DLC depends on the data field that carries the actual message to the respective destinations. DLC value records the number of bytes the data field is carrying which can be a maximum of 8 bytes long. The rest of the fields are there to ensure smooth frame delivery. In this research, we will be making use of the arbitration ID and the data part of the CAN frame.

2.1.1 Security issues of the CAN bus

CAN has several vulnerabilities that mostly arise from its design characteristics. The real-time communication requirements hinder the CAN bus from incorporating security measures in its communication. We want our car to stop immediately when we press the brake pedal; it would be disastrous otherwise. This makes the CAN bus vulnerable to insertion attacks, as none of the frames coming to the CAN would be authenticated. Moreover, non-authenticated systems can be vulnerable to fuzzy attacks. Attackers can cause unintended vehicle behavior by sending messages of randomly spoofed identifiers with arbitrary data [27]. The CAN network is not segmented. An attacker might compromise one type of network to take control of the network with safety-critical ECUs. CAN is also vulnerable to DoS attacks due to its design scheme in collision detection through carrier-sense multiple access with collision detection (CSMA/CD). When two arbitration IDs appear in the CAN bus, the resource is prioritized for the frame with a lower arbitration ID backing off the frame with a higher ID. By flooding the CAN bus with a high-priority CAN frame, attackers can quickly exploit this attribute and launch a DoS attack. An ECU's frame is broadcast to all other ECUs connected to the network. Each connected ECU decides whether it should act upon the received frame or not. This can be taken advantage of by simply eavesdropping on the CAN bus to learn the characteristics of CAN frames [25].

So far, we have discussed attacks that make use of the CAN bus design features. The other types of CAN attacks are those that are based on the generation of errors. Considering an attacker has remotely controlled a node, he might have the capability of forcing the ECU node to a bus-off state by creating a collision with multiple frames having the same arbitration IDs [28]. In this research, we aim to detect all the attack types regardless of the attack target.

2.1.2 CAN attack surfaces

Attackers should always find a way to send their attack packets to the internals of the vehicle so as for them to manipulate the network of vehicles. According to [12], attackers might gain access to a car's internals in two ways. The first is to physically approach the target vehicle and insert a malicious component

into a car's internal network via the ubiquitous OBD-II port or tapping into the CAN bus network. This can be easier to do but doing in so in bulk can be expensive. The other is through the various wireless interfaces available in present-day vehicles which is more complicated and multi-layered [29, 30, 31]. It can be by first intruding into the driver's phone that the driver might later connect it to their vehicle for entertainment purposes.

2.2 Related works

Over the past few years, the automotive industry has evolved from a blind and disconnected system to a connected system that can sense the surrounding environment [32]. While these advancements in technology have made our lives easier, they pose a security risk. As a countermeasure for the security risks involved, attack prevention and detection mechanism has been studied.

Network segmentation, encryption, and authentication on top of the vulnerable CAN bus can help in preventing attacks in in-vehicle networks. Such techniques will prevent attacks by automatically triggering predefined policies against unidentified threats [33, 34]. Network segmentation helps secure the CAN bus by implementing subnetworks. The segmentation over the subnetworks provides control over who can access a subnetwork. [35] applied network segmentation approach. Authentication and encryption are also among the techniques used to prevent attacks on the CAN bus. In [36], replay attacks have been avoided by an authentication method that is created using combination of SHA3 and HMAC function that expire with a session keys. Other approaches that use the technique of authentication include [24, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47]. [48] implemented an algorithm that encrypts the 8 bytes CAN frame using a symmetric key. They have managed to secure the CAN bus with synchronized key generators that can change the key dynamically. [49] has also brought encryption and authentication to the CAN bus. The method exploits the use of the stream cipher RC4 to encrypt and authenticate each CAN message. [50, 51, 52, 53, 54, 55] are also among the research works which apply encryption for secure CAN bus communication. One drawback of implementing encryption and authentication methods in the CAN bus is the real-time communication requirements.

Another way to secure the CAN bus is through anomaly detection approaches. The goal of anomaly detection is to establish a notion of normality that describes most of a given dataset. Thereafter, deviation from this normality of any sort will be detected as an anomaly [56]. It involves the use of CAN bus information to detect attacks and take appropriate measures. The IDS in in-vehicle networks can be categorized into four classes: fingerprint-based, parameters-monitoring-based, information-theoretic-based, and machine-learning-based [29]. Fingerprints-based methods [57, 58, 59, 60, 61] methods take into consideration the fact that different ECU on the in-vehicle networks usually have unique hardware fingerprint information, like electric signals, and using this information it analyzes the change of these signals for intrusion detection. It is a physical level approach, but attacks can be bypassed if they are applied at the application level. Parameters-monitoring-based [62, 63, 64, 65, 66, 67, 68] methods collect different static values (frequency, mean, variance, etc) that will later be compared with predefined values for intrusion detection. These approaches also have the drawback of heavily depending on periodic packets and being ineffective for unknown security threats. The other two categories, information-theory-based and machine learning-based, come here to solve this issue. The information-theoretic-based method [69, 70] is based on the fact that malicious messages injected into the normal communication will affect the network stability, and the information entropy can reflect the anomaly. Even if this has a small computational overhead, it is mostly ineffective in attacks that modify the data portion of CAN packets. Usually, IDSs further train machine learning models by taking features from the statistical-based or fingerprinting-based methods to further improve the problem at hand.

In recent years, the introduction of new types of malware has been increasing at a staggering speed. It shows that we need an algorithm capable of coping with future unknown attacks because signature-based approaches only detect known attacks. Machine-learning algorithms work better at detecting unknown attacks. [71] proposed a classification-based support vector machine that classifies benign and anomalous CAN frames. Similarly, [72] also uses deep neural networks for classifying attack packets. They have trained a Deep Neural Network (DNN) that is capable of identifying intrusions. The features they used to train the DNN are extracted directly from the CAN packet by counting an occurrence of "1" in the

bit-field "DATA" field of a packet. Another deep learning technique is proposed by [73]. It is based on a time series prediction method called Long Short-Term Memory (LSTM) networks. [20] uses CAN signals reverse-engineered from the proprietary CAN bus data to train a multi-dimensional LSTM. Besides LSTM, CNNs have also shown tremendous success in various applications. Apart from solving the known image classification problems, CNNs have been used on the general Internet for malware classification [74, 75, 76, 77]. As an extension of this work, CNNs have been used for in-vehicle network classifications. [19] proposed a Generative Adversarial Network (GAN) based IDS that trains on images generated from the extracted arbitration IDs. These same researchers have improved the GAN-based performance by proposing a new inception-ResNet-based IDS [17]. To train their model, they extracted arbitration IDs from CAN packets. The arbitration IDs, which are in a hexadecimal format, are later changed to binary form. The binary form of 29 consecutive arbitration IDs is then used to create an image with a dimension of 29×29. They managed to correctly classify most of the attack packets and benign packets into the correct classes.

To enhance intrusion detection on the CAN system with machine learning, we must first identify the resources from which features will be collected. Dumping CAN data from the in-vehicle network gives the timing information, arbitration IDs, and data. The timing information is usually used to just keep the sequence of frames. In our first work in this thesis, arbitration ID sequences are analyzed to improve a previously studied work. [15], was the first work in this category when our publication was made. It trains a single transition matrix that will be used to test the possible transitions between two different arbitration IDs. Our work has improved on this work by implementing an LSTM system that learns the sequence of arbitration IDs.

The LSTM-based ID sequence predictor works in most frequent arbitration IDs, but it fails in learning the sequence of less frequent and non-periodic arbitration IDs. It has also been improved by a CNN-based approach in [17]. The images used to train the CNN in this work are generated from a sequence of arbitration IDs. A simplified Inception-ResNet model is then trained using the images. However, the images generated in this method do not grasp the temporal dependency in the sequence of arbitration IDs. Improving this cutting-stage research would require

us to handle the temporal dependency of arbitration IDs. We did so in our work titled Rec-CNN using recurrence plots [18] that is capable of showing temporal information in images.

If only arbitration ID sequences are used to train machine learning models, no matter how good the sequence is learned, the models will not detect attacks that alter the data part of CAN frames without altering the sequences of IDs. Using the data part of the CAN frame can also be difficult as it contains multi-dimensional data. In [73], the method proposed needed to implement a single architecture for all the available arbitration IDs. In [20], they handled the high-dimensional CAN bus data but their method requires reverse-engineering of the CAN bus data. Our research comes in handy in solving these two issues, avoidance of reverse-engineering and handling high-dimensional CAN bus data. We wanted to avoid the reverse-engineering part so as for the IDS system to work in all types of vehicles independent of the make and model. And the handling of the high-dimensional CAN bus data is required so that this IDS system could easily be optimized as part of other IDS systems. Similar to LSTM-based arbitration ID sequence analysis, training LSTM using CAN data can be challenging due to the high number of features and LSTM hyperparameters. We chose CNN over LSTM due to this reason. Current studies have already applied CNNs to detect CAN intrusions, but the features used are extracted from either only arbitration IDs or raw CAN data. Our last proposal called U-CAN uses a hamming distance (HAMD) distribution of CAN frame bits to extract features. It can also be used with reverse-engineered CAN frames.

3 Sequential data analysis for CAN bus intrusion detection: LSTM-IDS

In-vehicle network attacks often disrupt the regular flow of data inside the network. Studying the flow of data inside in-vehicle networks can be one approach to identifying such attacks. This chapter introduces the use of LSTM to learn the sequence of arbitration IDs in the CAN bus. If LSTM can learn the sequence of arbitration IDs, it can detect any in-vehicle network attacks that deviate from the normal sequence of the IDS. In this chapter, we have explained in detail how LSTM can be used for detecting in-vehicle network attacks.

3.1 Introduction

To improve the security flaws of the CAN bus, in this section we have devised an IDS by analyzing the sequence of CAN frames' arbitration IDs. In addition to other information, CAN frames have an arbitration ID that is used to control the priority of CAN frames. Our intrusion detection method extracts the IDs to train Long Short-term Memory Networks (LSTM). Once, the LSTM network learns the pattern of the IDs, if there is any, it is used to predict an arbitration ID that might appear after a certain sequence of arbitration IDs. Relying on the predicted arbitration ID, an anomaly signal is prepared from a Softmax probability of all the available classes (all the arbitration IDs). An anomaly is detected using two ways. The first approach compares the prediction probability of all the classes and selects the one with the highest probability as a predicted arbitration ID. The predicted ID is then compared with the true ID for anomaly detection. The

second approach gets an aggregated categorical cross-entropy loss of the predicted classes that will be later compared with a predefined anomaly threshold to detect intrusions. The transition-matrix-based method proposed by [15] trains a single transition matrix that will be used to test the possible transitions between two different IDs. Even if this performs with a near-perfect precision value, it has a very low recall value as it is impossible to grab millions of arbitration ID sequences in a single transition matrix. In this section, we proposed an IDS system using LSTM that greatly outperforms the transition-matrix-based method.

The main contribution of this research study is summarized as follows:

- We have proposed an intrusion detection system using LSTM. The LSTM is trained on the sequence of CAN frames arbitration IDs.

- Two ways of anomaly detection methodologies are proposed. The first compares LSTM predicted arbitration ID with true ID for intrusion detection. That is, an attack is flagged if the predicted and true IDs are not the same. The second is using a categorical cross-entropy loss. For a batch of data, the loss is calculated from the LSTM's Softmax output and one hot encoded true label. This loss is then compared with a predefined threshold for intrusion detection.

- Our LSTM-based IDS outperforms the first work that studies the sequence of IDs [15] for intrusion detection in RPM (revolution per minute) attacks and gear attacks.

3.2 Long short-term memory networks

LSTM was proposed to avoid the vanishing and exploding gradient problems in Recurrent Neural Networks (RNNs) so as for it to learn to bridge time intervals over 1000 steps without loss of short time lag capabilities. Figure 3.1 shows a block diagram of the LSTM network. Unlike conventional RNN, LSTM has LSTM cells that have an internal recurrence gates that are updated according to the forward propagation equations shown in equations 3.1 to 3.5.

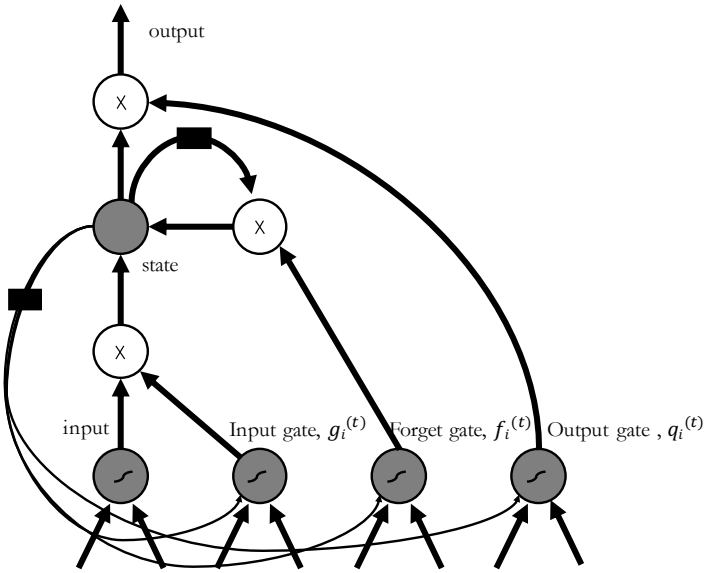


Figure 3.1: Block diagram of the LSTM recurrent network cell

The most important is the state unit $s_i^{(t)}$, which has a linear self-loop that is controlled by forgetting gate unit $f_i^{(t)}$ (for time step t and cell i), equation 3.1, which sets the self-loop weights to a value between 0 and 1 via the sigmoid unit.

$$f_i^{(t)} = \sigma(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}) \tag{3.1}$$

$x_{(t)}$ is the current input vector and $h_{(t)}$ is the current hidden layer vector, containing the outputs of all the LSTM cells, and b^f, U^f, W^f are respectively biases, input weights, and recurrent weights for the forget gates. The LSTM cell internal state is thus updated using (2), but with a conditional self-loop weight $f_i^{(t)}$.

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}) \tag{3.2}$$

b denotes the biases and U and W are input weights and recurrent weights respectively. The external input gate unit $g_i^{(t)}$ is computed similarly to the forget gate (with a sigmoid unit to obtain a gating value between 0 and 1), but with its parameters as in equation 3.3.

$$g_i^{(t)} = \sigma(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}) \tag{3.3}$$

The output, $h_i^{(t)}$, of the LSTM cell can also be shut off, via the output gate, $q_i^{(t)}$, which also uses a sigmoid unit for gating. The equations used to calculate $h_i^{(t)}$ and $q_i^{(t)}$ are shown in equations 3.4 and 3.5 respectively.

$$h_i^{(t)} = \tanh(s_i^{(t)} q_i^{(t)}) \tag{3.4}$$

$$q_i^{(t)} = \sigma(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}) \tag{3.5}$$

Parameters b^o,U^o,W^o are for its biases, input weights and recurrent weights respectively. Once all these parameters are updated, there is a need for backpropagation that is calculated from the selected loss of the LSTM. The loss calculation is similar to RNN and it can be referenced from [78].

3.3 Input data preprocessing

Two datasets are used in the experiment. The initial dataset was collected from a Hyundai's YF Sonata car and is available for use at [19]. Throughout this section, it is referenced as RawHyu_data. The second dataset was collected from a public passenger vehicle, however owing to privacy considerations, we are not going to publish it here. This dataset will be referenced as PRV_data.

The input to the LSTM network is only a sequence of IDs. Like all types of neural networks, LSTM also accepts only numeric tensors. Each arbitration ID is tokenized to convert the sequence of the IDs to a numeric tensor. After tokenization of each arbitration ID to a numeric value, the sequence of numbers is one hot encoded before it is fed to the network. Figure 3.2 shows the RawHyu_data's list of arbitration IDs and corresponding frequency in a sequence of 5000 frames. For an arbitrary sequence length, seq_len, the one-hot encoded input to the network will have a shape of $[seq\_len, 27]$. Similarly, Figure 3.3 shows the PRV_data's

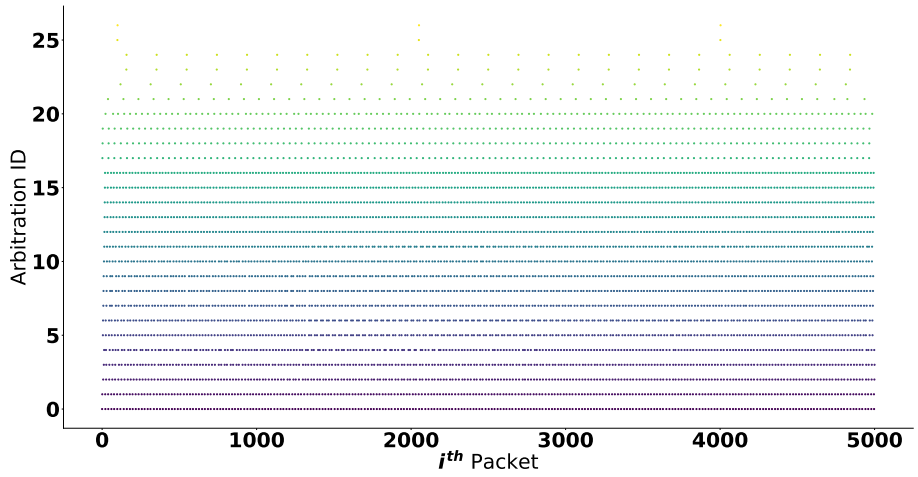


Figure 3.2: RawHyu_data Scattered plot of the first 5000 frame sequences of all the 27 arbitration IDs.

list of arbitration IDs and corresponding frequency in a sequence of 5000 frames. One hot encoded shape of the input will be $[seq\_len, 42]$ for this dataset as it has 42 unique arbitration IDs.

3.4 Proposed method

3.4.1 IDS network architecture

The input to the neural network is the sequence of IDs extracted from CAN frames. The network architecture of this IDS is dependent on the target dataset (RawHyu_data, PRV_data). In this section, the general skeleton of the network architecture is presented excluding the hyperparameters considered in the hyperparameter search algorithm. The network consists of 6 layers in the sequence dense, dense, dropout, LSTM, LSTM, and dense. The two top dense layers are ReLu (Rectified Linear Unit) activated and the node size in both layers is searched from Dense 1 nodes and Dense 2 nodes of Table 3.1. Next, there is a dropout layer that is also searched using the hyperparameter search algorithm from the list of values in Table 3.1. The output from the dropout layer is then fed to two LSTM layers that are both tanh (Hyperbolic tangent) activated with a dropout

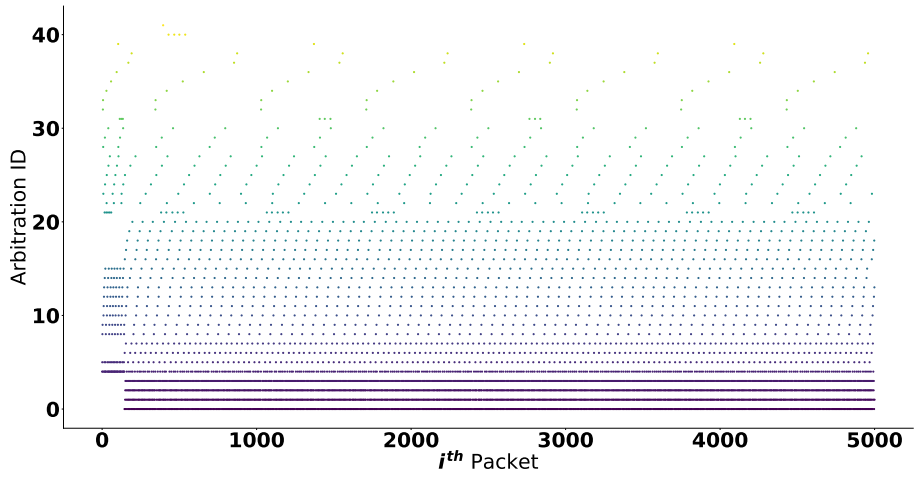


Figure 3.3: PRV_data Scattered plot of the first 5000 frame sequences of all the 42 arbitration IDs.

value of 0.2. Two LSTM layers are stacked in the architecture and doing so would require us to return the sequence hyperparameter to true in the first LSTM layer. Likewise, the first LSTM layer in the network architecture is returned to the second LSTM layer. The number of nodes in both the LSTM layers is considered in the hyperparameter search. The dimensions of these nodes are shown in LSTM1 nodes and LSTM2 nodes in Table 3.1. The general network is shown in Figure 3.4. As shown in the figure, the top layer has a dense layer with a sigmoid activation function that gives a sigmoid probability of the available classes. The node size in this layer is the same as the number of classes (arbitration IDs) in the target dataset. The number of nodes is 27 for RawHyu_data and 42 for PRV_data each representing the number of unique arbitration IDs in the dataset.

The arbitration ID prediction process for the CAN bus of PRV_data is shown in Figure 3.5. Depending on the shape of the input selected, seq_len, the process first collects seq_len arbitration IDs from the CAN bus. Using this sequence of IDs as an input to the model, it outputs the Softmax probability of all the available classes.

The one given with the highest probability will be the predicted arbitration ID. Next, we compare the predicted arbitration ID with the one that has appeared after seq_len arbitration IDs. If the predicted and true arbitration IDs are not

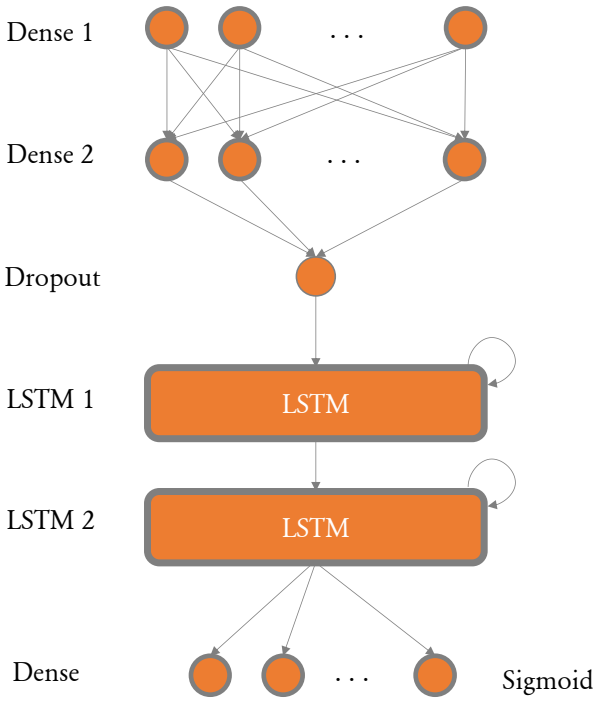


Figure 3.4: ID prediction network architecture

the same, an anomaly signal is flagged. But, if both of these values are the same, we update the input and the true values in the next step. The input value will be a tensor that grabs seq_len arbitration IDs again but this time the start pointer is updated by one to point to the ID next to the first one. And the last pointer will also be incremented by one to incorporate the last predicted arbitration ID. Using these arbitration IDs we again go through the same process to monitor for intrusions. This process starts from when the engine of the car is started and continues till the car's engine is stopped.

3.4.2 Hyperparameter search

LSTM has a lot of parameters that need to be set to an optimal value. Setting these hyperparameters is usually done using hyperparameter search algorithms. For the thesis, we mainly used Hyperband [79] which speeds up hyperparameter settings using a random search algorithm. For both datasets, we searched through the set of hyperparameters listed in table 3.1. In the table, sequence length refers

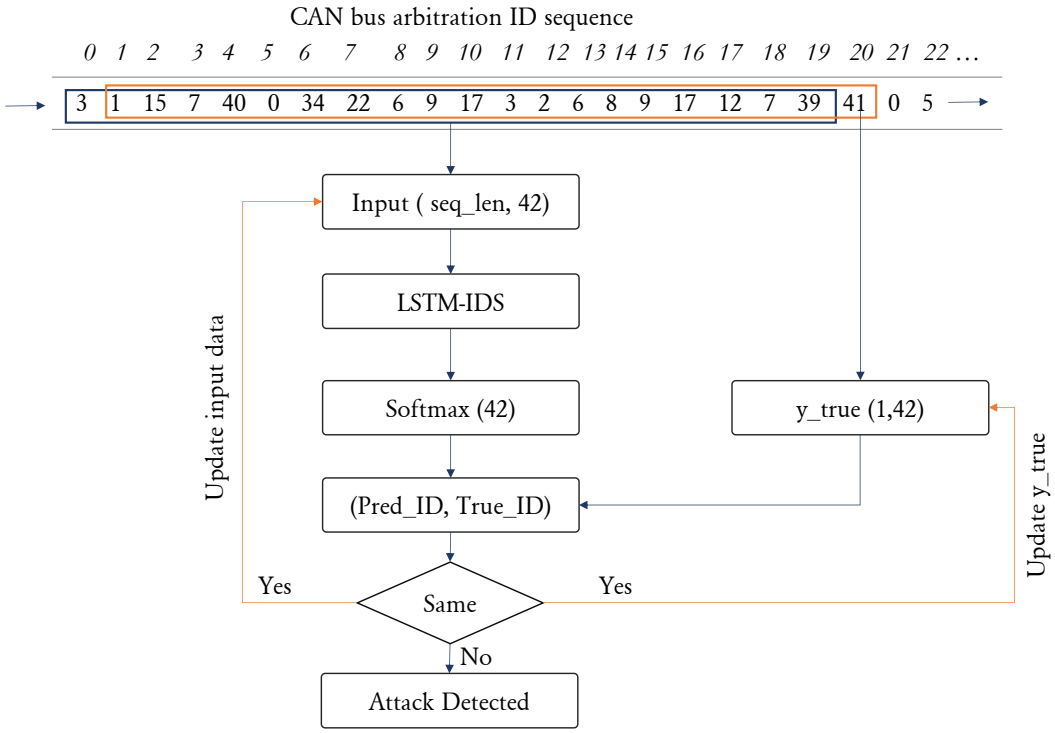


Figure 3.5: ID Sequence Prediction IDS process

Table 3.1: hyperparameter values used for searching the LSTM network architecture

Hyperparameter type	**List of values**
Sequence length	20, 40, 80
Batch size	32, 64, 128
Dense 1 nodes	32, 64, 128, 256
Dense 2 nodes	32, 64, 128, 256
Dropout	0.1, 0.5, 0.9
optimizers	Adadelta, Adagrad, Adam Adamax, Ftrl,Nadam RMSprop, SGD
LSTM 1 nodes	128, 256, 512
LSTM 2 nodes	128, 256, 512

to the total number of arbitration IDs LSTM looks back to make a single ID prediction. The rest of the hyperparameters are self-explanatory.

3.4.3 Attack frames and anomaly signal

PRV_data is collected from a public passenger vehicle and it only consists of a normal sequence of arbitration IDs. In this dataset, we will only be showing how LSTM can learn the sequence of the IDs. In doing so, we will show if LSTM can detect attacks that alter the ID sequences. RawHyu_data has both benign and attack data. LSTM in this dataset will be trained on the attack-free dataset. It is then used to detect attacks in the attack dataset. The attacks are targeted at "spoofing the gear" and "spoofing the revolution per minute (RPM) gauge" of the vehicle.

The IDS system uses two approaches. The first approach simply takes the last layer's Softmax output. The class with the highest probability will be the predicted arbitration ID. Attacks will be flagged in this stage if the prediction doesn't match the actual class label. This approach requires us to go through each arbitration ID which can be resource-consuming. The second approach is based on the loss value of predictions in a batch of data. The model takes any batch size and makes a prediction at a single run. Then categorical-cross entropy is used

to evaluate the loss in this batch. Categorical cross-entropy loss is used because it penalizes higher errors than low errors. In a batch of data, it is calculated from the Softmax probability output of the model and the one-hot encoded true labels. Let the true labels for our predicted arbitration ID be encoded as 1-of-K binary indicator matrix Y, i.e.,$Y_{i,k} = 1$ if sample i has label k taken from the set of K arbitration ID labels. Let P be a matrix probability estimates, with $p_{i,k} = Pr(t_{i,k} = 1)$. Then the log loss of the whole set is calculated by using Equation, 3.6.

$$L_{\log}(Y, P) = -\frac{1}{N} \sum_{i=0}^{N-1} \sum_{k=0}^{K-1} (y_{i,k} \log P_{i,k}) \tag{3.6}$$

Some arbitration IDs appear to be less frequent than others. Due to prediction errors, such IDs may result in a greater loss value. To circumvent this issue, the overall cross-entropy loss is evaluated in consideration of the ratio of arbitration ID frequencies. This gives the loss value for frequent arbitration IDs a larger weight than for less frequent IDs.

3.4.4 Training and test set description

The RawHyu_data is publicly available data with 988,871 sequences of arbitration IDs. Out of this data, we used 70% for training, 15% for validation, and the last 15% is used as testing data. The purpose here is if the model can accurately predict the sequence of arbitration IDs. If it can do so, it can detect any kind of attacks that may disturb the correct sequence of arbitration IDs. The last 15% of the dataset will show if the model is capable of doing so. PRV_data is a sequence of 1,360,000 arbitration IDs obtained from a public passenger vehicle. Same as RawHyu_data, this is also divided into 70%, 15%, and 15% for training, validation, and testing, respectively.

3.5 Experiment results and discussion

The experimental results of the methodology are described in this section. In the beginning, we demonstrate how the neural network's hyperparameters are

Table 3.2: Hyperband search results

Hyperparameter type	**List of values**
Sequence length	40
Batch size	32
Dense 1 nodes	64
Dense 2 nodes	64
Dropout	0.1
optimizers	RMSprop
LSTM 1 nodes	512
LSTM 2 nodes	256

tweaked. The creation of an anomalous signal using Softmax or aggregated loss is then accomplished using the chosen hyperparameters, as is further explained in the sections that follow. This section goes into great length to discuss the findings from utilizing both anomaly signal methods.

3.5.1 Hyperparameter search results

The hyperparameters of neural networks are among the most important factors that must be fine-tuned to achieve the best outcomes. In this work, a hyperparameter search algorithm called Hyperband is used to fine-tune the hyperparameters. Table 3.2 shows the search results for the hyperparameter values listed in Table 3.1. RawHyu_data is used to search the hyperparameters. These similar values are then applied to train the PRV_data to see if these hyperparameters can be generalized for a different kind of data. For better performance, one needs to do a hyperparameter search for each make and model of a car.

3.5.2 Softmax-based anomaly detection

The Softmax-based approach works by hard coding an ID prediction neural network. The approach tests for anomalies by considering an arbitrary sequence of arbitration IDs. And a prediction is selected by taking the one with the highest probability from the Softmax output of the last layer in the model. Table 3.3

Table 3.3: Softmax based ID prediction results

Level	RawHyu_data	PRV_data
Top-1	0.894	0.893
Top-3	0.975	0.965
Top-5	0.988	0.984

shows the arbitration ID prediction accuracy for RawHyu_data and PRV_data. In the table, top-1 refers to the conventional way of calculating accuracy by considering a class with the highest probability as a predicted class. Top-3 and top-5 consider the top 3 and 5 prediction probabilities, respectively. A correct class is taken as correctly classified if the class appears in the top 3 or top 5 of the highest probabilities.

Figures 3.6 and 3.7 show confusion matrix results in a heat map graph. As it can be seen in the figures, the models fail to correctly classify the less frequent arbitration IDs. These IDs are in the lowermost corner of the graph. Due to this, we devised a second approach that calculates anomaly signals using log loss of the predicted and true classes.

3.5.3 Loss-based anomaly detection

In this method, a categorical cross-entropy loss is used as an anomaly signal. The loss value of the Softmax output and the true value of a batch of data multiplied by the ratio of arbitration ID appearance in the CAN bus is used. The ratio of arbitration IDs is added so as for the most frequent arbitration IDs to have more effect on the loss values than the less frequent IDs. In such a way, "spoofing the gear" attack, and "spoofing the revolution per minute (RPM) gauge" attacks released with RawHyu_data have been tested. In the dataset, '041f' is the arbitration ID for gear and '0316' is the arbitration ID for RPM. Figures 3.8 and 3.9 show the scatter plot of loss values of attack and non-attack windows. The horizontal dotted line represents the attack detection threshold. Any loss value above the boundary is classified as an attack; otherwise, it is considered benign. Minding the results shown in figure 3.6 for these two arbitration IDs, the scattered plots also show similar values. The gear attack is better detected than

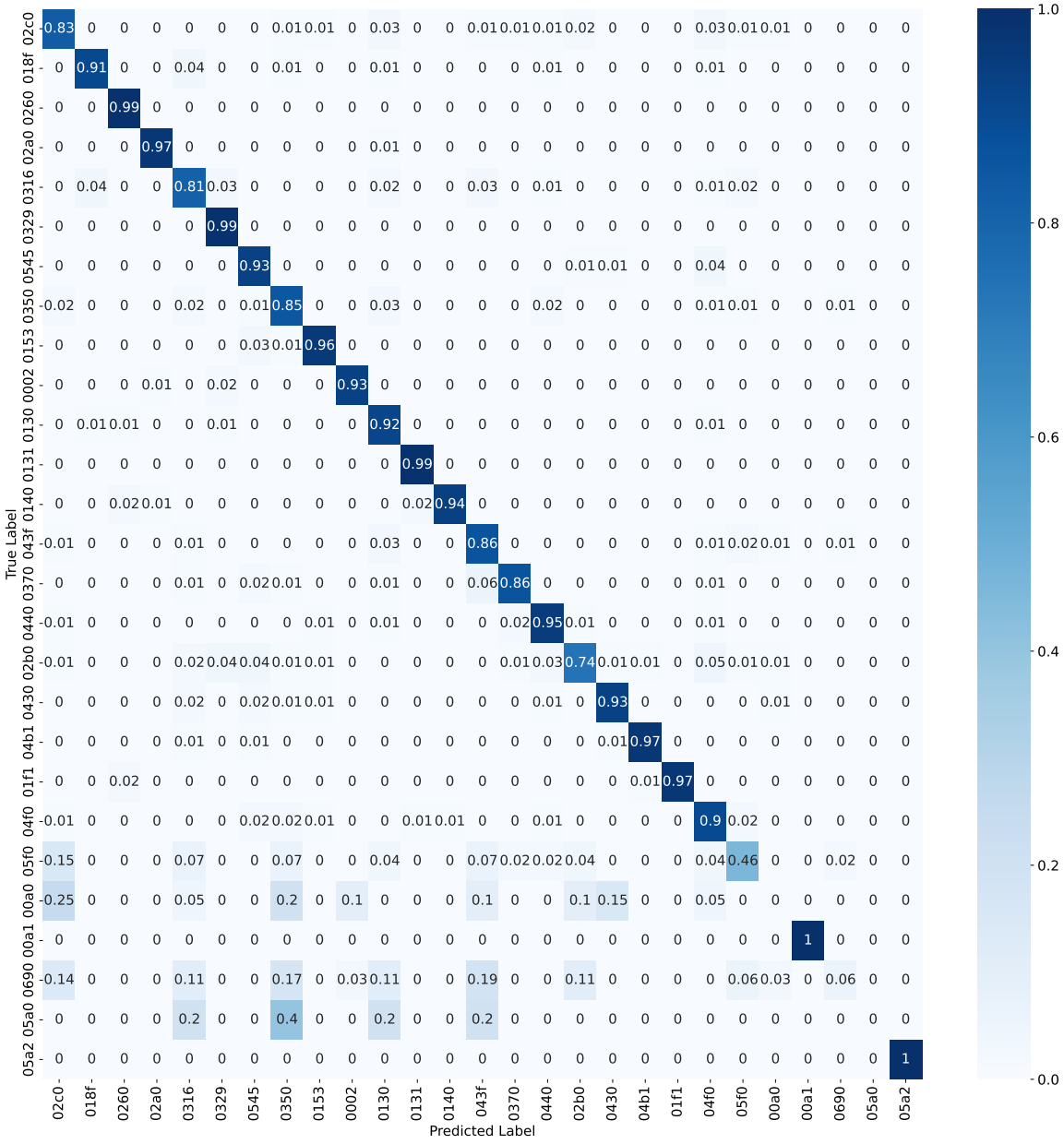


Figure 3.6: Confusion matrix for RawHyu_data

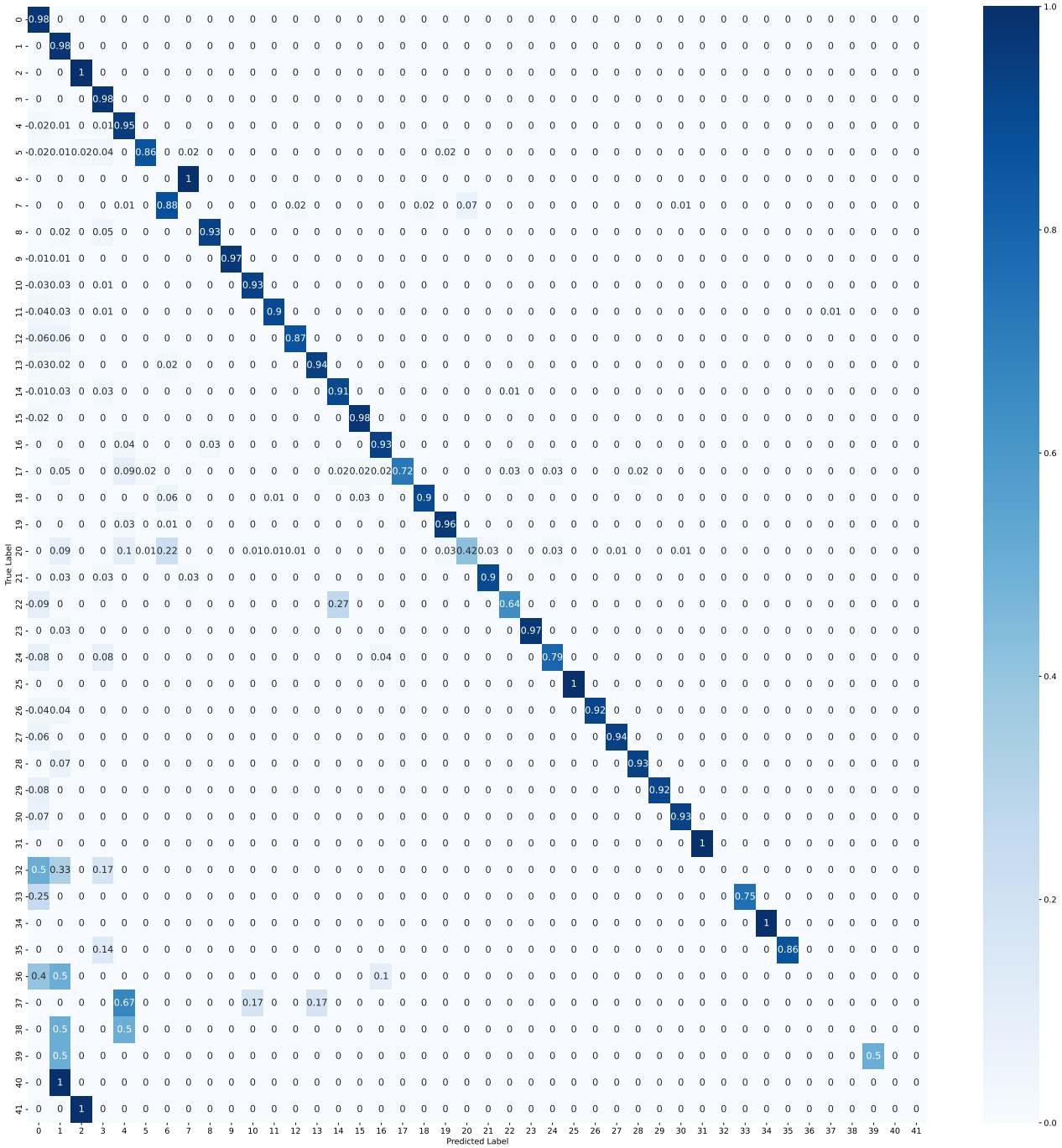


Figure 3.7: Confusion matrix for PRV_data

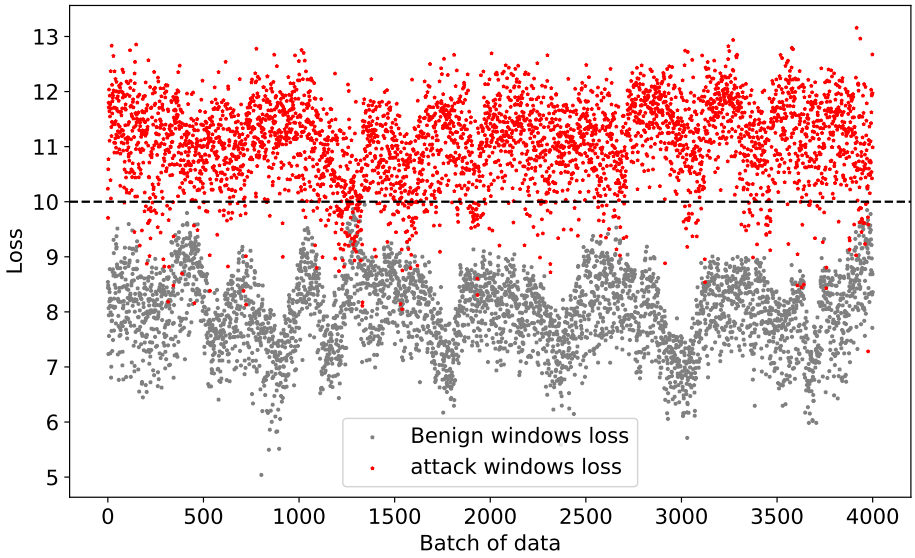


Figure 3.8: Loss based gear attack detection

Table 3.4: Confusion matrix for gear attack detection

	Attack	**Benign**
Attack	0.909	0.091
Benign	0.0002	0.9998

the RPM attack as the sequence prediction model better predicts gear IDs than RPM IDs.

Tables 3.4 and 3.5 show the confusion matrix of attack detection for gear and RPM attacks, respectively. The model prediction performance for RPM (0.81) is lower than the gear (0.86). This small difference has created a big gap in detection performance as shown in the tables. This shows us that for the other IDs with higher prediction accuracy, attacks will be better detected.

The experimental result of this work is compared with the transition-matrix-based method that uses a transition matrix for ID sequence prediction. As it is shown in figure 3.10, the F1 score of the proposed method outperforms the transition-matrix-based method.

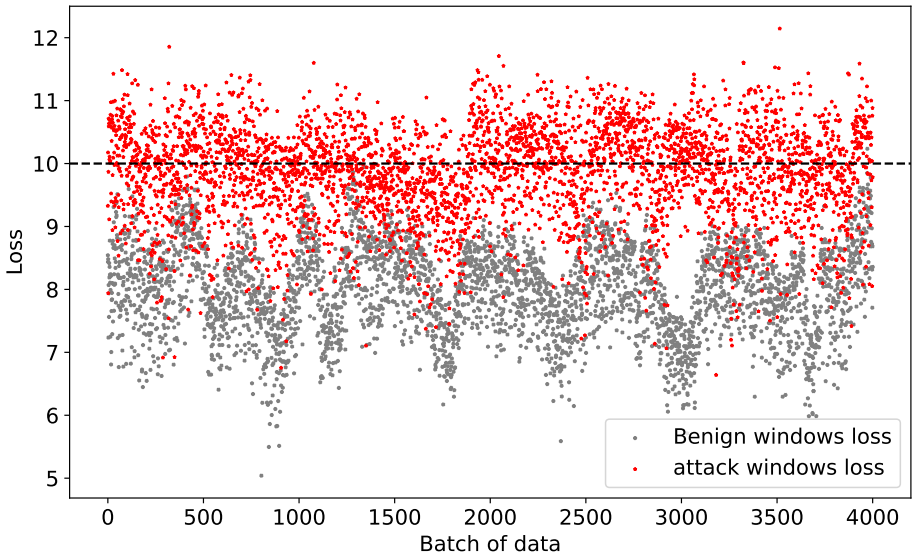


Figure 3.9: Loss based RPM attack detection

Table 3.5: Confusion matrix for RPM attack detection

	Attack	**Benign**
Attack	0.460	0.540
Benign	0.0002	0.9998

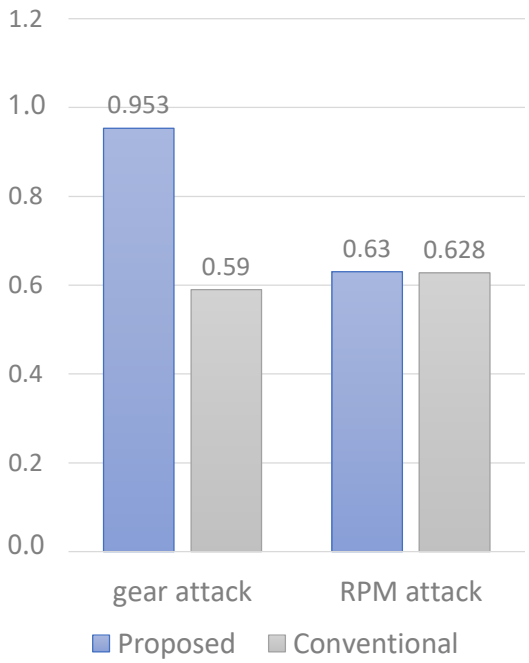


Figure 3.10: F1 score comparison of the proposed and transition-matrix-based methods

3.6 Conclusion

In this chapter, an intrusion detection system using LSTM based is proposed by analyzing the sequence of arbitration IDs. The experiment focused on intrusion detection in in-vehicle networks, but the idea can be extended to anomaly detection of other types of sequential data. The model is based on two approaches. Once the LSTM network is trained, the first approach uses the highest Softmax probability to select the next arbitration ID. The predicted arbitration ID is then compared with the true ID for detecting anomalies. The second approach is an improvement to the first one by using a categorical cross-entropy loss anomaly signal. The Softmax probability for each arbitration ID is used to calculate the log loss of the predicted ID and the true ID. The log loss is then compared with a predefined threshold for intrusion detection. The first approach can be impossible for real-world use because single ID predictions will take longer but the experimental results from the second approach show that our model can be implemented in a real vehicle. Attacks that can be detected using these approaches are the kind of

attacks that might alter the normal sequence of arbitration IDs. However, attacks that don't alter the sequence (e.g. impersonation attacks) will not be detected using these approaches. In the subsequent chapters, we have incorporate more features from the data sequence so as for our system to identify these kinds of attacks and improve the detection performance of the selected attack types.

4 Application of Image classification in the CAN bus intrusion detection: Rec-CNN

This chapter is also an improvement to the previous chapter's arbitration ID sequence analysis. LSTM might be the best sequence learning algorithm but it didn't capture the sequence of arbitration IDs accurately. As a result, We proposed a CNN-based intrusion detection system that has improved performance than the LSTM-based method.

4.1 Introduction

The methodology that is proposed in this chapter takes advantage of advancements in deep neural networks for intrusion detection. Deep learning has shown promising results in various fields, especially in image detection and classifications most notably, CNNs [16]. In this work, we show how CNN's performance in images could solve intrusion detection problems. To do so, we need to change the intrusion detection problem to an image classification problem. Previous research has employed CNNs for in-vehicle network intrusion detection [17]. It builds a 2D grid data frame from arbitration IDs as an input to the deep CNN network. Each CAN frame has an arbitration ID that is used in the CAN bus to resolve collisions through a bit-wise arbitration. The generated images are then trained using a simplified Inception-ResNet model for in-vehicle network intrusion detection. However, the images generated in this method do not grasp the temporal dependency in the sequence of arbitration IDs in the CAN bus. As far as our knowledge, this is the only research that uses CNNs trained on arbitration IDs for in-vehicle network

security. We will report a comparison to this state-of-the-art Inception-ResNet-based method throughout our work.

Improving the Inception-ResNet-based method would require an image generation algorithm that can also keep the temporal dependency. We have solved this issue using recurrence plots to create an image from the CAN bus's time-series data, hence Rec-CNN. We first collect packets that have arrived on the CAN bus; these packets contain signal information and arbitration IDs. The arbitration IDs are used to control the priority of using CAN bus resources with lower arbitration IDs having a high priority. To generate the images, we first extracted these arbitration IDs in a sequence. With this sequence of arbitration IDs, a square matrix is created using a recurrence plot algorithm, but unlike normal recurrence plots [18], we have used categorical recurrence plots. The main reason for this is for plots to show no special effect to closer encoding of arbitration IDs than the further encoding of arbitration IDs. The formal operation of recurrence plots is performed if and only if the two arbitration IDs are encoded similarly. Using these images of recurrence plots, we have experimented on how CNNs can easily be trained to classify attack and benign sequences of arbitration ID for a secure CAN bus communication.

The main contributions of this work are as follows:

- To the best of our knowledge, this is the first work that uses recurrence plots to generate images out of in-vehicle network data. Using recurrence plots helps us show the temporal dependency of a sequence to images.

- We have shown how a simple two-layered CNN with carefully selected hyperparameters can outperform state-of-the-art research in this field. We have selected the hyperparameters using a hyperparameter search algorithm called Hyperband.

- We have trained and tested the proposed model with a publicly available dataset and a dataset we have collected from our target vehicle. Our experiments prove the better performances of the proposed method over the state-of-the-artwork based on Inception-ResNet [17] in both datasets.

We have used two types of datasets to train Rec-CNN, simulated attack datasets,

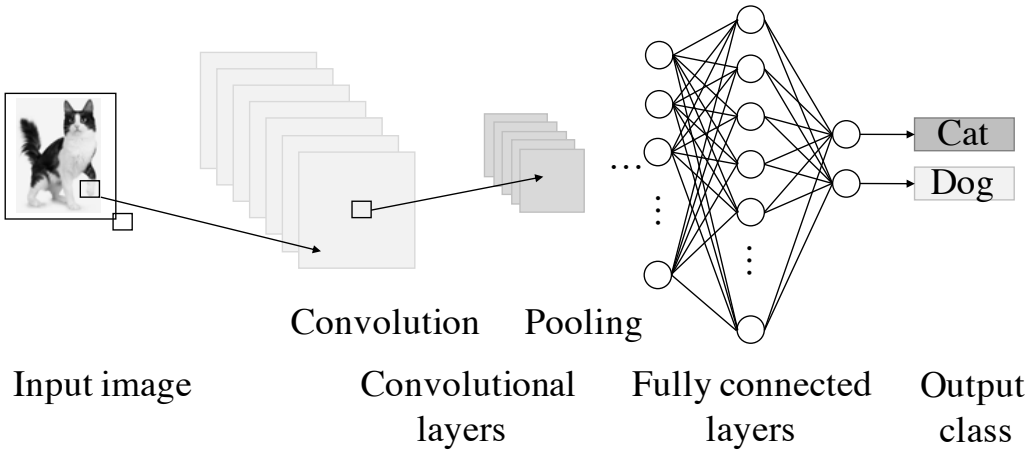


Figure 4.1: A general CNN architecture for image classification

and real-world datasets. In the next section, we explain in detail the steps we followed to simulate CAN attacks.

4.2 Convolutional neural networks

Inspired by the study of the brain's visual cortex, convolutional neural networks (CNNs) have been used for image recognition and classification problems since their inception in 1980 [16]. There are various types of CNNs, but in general, CNNs consist of convolutional and pooling layers, which are grouped into modules. Depending on the problem at hand, one or more standard fully connected neural networks are added on top of these modules. The top layer is mostly used for predicting the class label with some activation functions [80].

Describing each interaction between input units and output units in artificial neural networks (ANNs) can be prone to over-fitting and expensive to very deep architectures. The motivation for using convolutional layers comes out of this. Convolutional layers of CNNs have sparse interactions, which is accomplished by smaller kernels than the input. On top of that, CNNs, unlike ANNs, are capable of sharing parameters that make CNNs faster in operations. This property of parameter sharing causes the layers to have an equivariance property to translation; when the input changes, the output changes in the same way [81].

A general CNN architecture looks like Figure 4.1. The convolutional operation

in the figure refers to the computation of the input image with a weighted kernel that revolves in the input image. Equation 4.1 shows this operation, for an input image I, and a kernel K with an $m \times n$ dimension. The resulting pixel value at row i and column j would be $S_{i,j}$.

$$S_{(i,j)} = (I * K)(i,j) = \sum_{m} \sum_{n} I(m,n)K(i-m, j-n) \tag{4.1}$$

We still need to discard irrelevant information from the convolutional operation done by the pooling layer. The pooling function does so by replacing the previous layer's output with a summary of the nearby outputs. The two most common pooling layers are the max pooling and average pooling layers. The max-pooling layer propagates the maximum value within a receptive field to the next layers. The average pooling layer takes the average of all the values in the receptive field. Equations 4.2 and 4.3 are the formulas for max pooling and average pooling of a region $R_{k,l}$, respectively, in the resulting matrix of $S_{i,j}$.

$$y_{zkl} = \max_{(p,q) \in R_{k,l}} S_{z(i,j)} \tag{4.2}$$

$$y_{zkl} = \frac{1}{|R_{k,l}|} \sum_{(p,q) \in R_{k,l}} S_{z(i,j)} \tag{4.3}$$

The fully connected layers are normal feed-forward network layers with or without activation functions. The output from the pooling layers is used as input for multiplication with each node in the layer. Training of the CNN, or deep learning in general, is done using backpropagation; for a detailed explanation on this topic, we recommend referring to [81].

4.3 Hyperparameter optimization

Hyperparameter optimization is the process of searching for an optimal set of hyperparameters of a machine learning architecture, CNNs in our case, that would result in better performance. Machine learning algorithms have no way of

learning these variables. Machine learning practitioners are expected to carefully select the parameters for a better-performing model. In practice, we can have different data types for these hyperparameters, including categorical, discrete, and continuous. Tuning these variables manually would be expensive for big machine learning models that take days or even weeks to finish. This has raised a new research direction that automates hyperparameter tuning called hyperparameter optimization. A few hyperparameter optimization algorithms are currently being used to automate this process. Babysitting, grid search, random search, gradient-based optimization, and Bayesian optimization is among the popular methods [82]. For our work, we have used Hyperband [79], which takes advantage of random search.

4.4 Recurrence plots

CNN has shown promising results in image classification, image detection, and image segmentation problems. To take advantage of these improvements, we needed to change our intrusion detection problem to an image classification problem. This is where recurrence plots (RPs) come into play to generate images out of time-series data. RPs are used to visualize recurrences in sequence and are expressed by a square matrix [18].

$$R_{i,j}(\varepsilon) = \Theta(\varepsilon - \|\vec{x}_i - \vec{x}_j\|),\ i, j = 1, ..., N \tag{4.4}$$

In Equation 4.4, N is the number of measured points $\vec{x}_i$, and ε is a threshold distance. Θ is a Heaviside function as defined in Equation 4.5, whose value is zero for negative arguments and one for positive arguments. $\|\|$ is a norm. The matrix obtained with the recurrence equation is projected to image pixels, with 1 representing a dark color and 0 representing a white color. In the resulting recurrence plot, the main diagonal will always be black because $R_{i,i} \equiv 1,\ i \in 1, ..., N$. This black line is called the line of identity.

$$\Theta(x) = \begin{cases} 0, & x < 0. \\ 1, & \text{otherwise.} \end{cases} \tag{4.5}$$

The most crucial parameter of RP is the threshold ε, but the RP we are employing in this research is categorical RP, and the threshold serves no purpose in this regard. Categorical RPs can use no meaningful distance metric to compare neighboring values. The values in categorical datasets are simply labels, and closer numerical labels have no special effect over further numerical labels. The pixel in the plot is flipped if and only if two classes have the same labels.

4.5 The proposed method: Rec-CNN

The proposed system is an IDS for in-vehicle networks. The general structure of the IDS system is shown in Figure 4.2. The system has three stages: preprocessing, training, and testing stages. The preprocessing stage deals with converting CAN frames to a datatype suitable for CNN model training. Section 4.5.1 describes in detail how a sequence of categorical frames is encoded and converted to a recurrence plot. In the training stage, we train the IDS model using the preprocessed stage results. Optimal hyperparameters are also selected using Hyperband. The details for this stage are available in sections 4.5.3 and 4.5.4. Finally, we need to deploy our trained model in a real-world environment. The testing stage does two types of inferences, online and offline, which are explained in section 4.6.

4.5.1 Datasets and feature engineering

We have tested the proposed method with two types of datasets. The first dataset is collected from our test vehicle, and the second is published with the Inception-ResNet-based method. These datasets are referred to as dataset A and dataset B, as mentioned earlier. The structure of both datasets is shown in Figure 4.3. In this research, we are using the timestamp and arbitration ID of the CAN packet to train our model. The timestamp is not directly used in training but is only used to keep the precedence of the packets. The arbitration ID of the dataset is extracted and encoded to a value between 0 and N, where N represents the total number of unique arbitration IDs. As shown in Figure 4.3, dataset A's arbitration ID is eight characters long, while dataset B's is only four characters long. This is because of the type of CAN network implemented in both test cars. Our test car runs on CAN 2.0-B whereas dataset B is collected from a car that runs on CAN

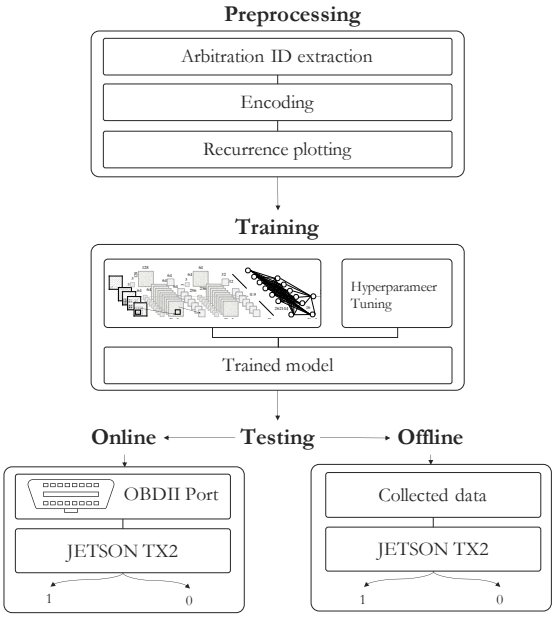


Figure 4.2: A graphical representation of Rec-CNN

2.0-A. CAN 2.0-B can have more arbitration IDs as it is 29 bits long. But, this did not affect our model as both datasets have a comparable number of arbitration IDs.

First, we collect 128 arbitration IDs of packets arriving on the CAN bus. The arbitration IDs are then encoded to a number between 0 and N, and the encoded sequence of arbitration IDs is converted to CNN's input image using RPs. The training image is subsequently generated out of the square matrix in the RPs with a dimension of 128×128. In the benign dataset, there are only N unique arbitration IDs and encoding starts from 0. But a new arbitration ID not found in the available arbitration IDs list might appear in the CAN bus. When these new arbitration IDs appear in the CAN bus, they are encoded as -1. For instance, the DoS attack in dataset B uses an arbitration ID of `0X000` to send high-priority packets, which we encoded as -1 whenever it shows up in the sequence.

Figures 4.4 and 4.5 show the RPs for the normal and all types of attacks in datasets A and B. In dataset A, the difference between the images for benign, drop attacks, fuzzy attacks, and insertion attacks look similar since there are few attack packets in the dataset. But in the case of dataset B, the images for

Timestamp	Interface	Arbitraion_ID	Data
(1557264444.790225)	can0	0CF00400#F4857D000000F07D	
(1557264444.792290)	can0	0CF00300#C1000000FFFFFFFF	
(1557264444.798501)	can0	18FEF100#C3000000001E00CF	

(a) Benign dataset A

Timestamp: 1479121434.850423 ID: 02c0 000 DLC: 8 14 00 00 00 00 00 00 00
Timestamp: 1479121434.850977 ID: 0430 000 DLC: 8 00 00 00 00 00 00 00 00
Timestamp: 1479121434.851215 ID: 04b1 000 DLC: 8 00 00 00 00 00 00 00 00

(b) Benign dataset B

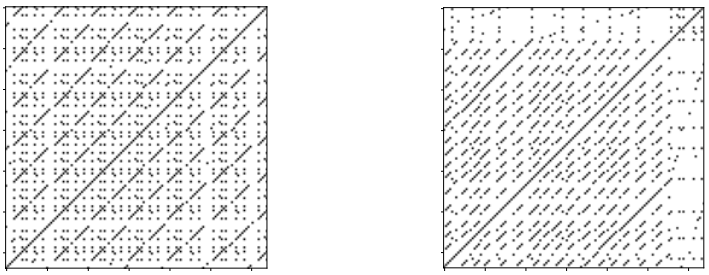
Timestamp	Arbitration_ID	DLC	Data	Label
1478198376.389427	0316	8	05,21,68,09,21,21,00,6f	R
1478198376.389636	018f	8	fe,5b,00,00,00,3c,00,00	R
1478198376.389864	0260	8	19,21,22,30,08,8e,6d,3a	T

(c) Attack dataset B

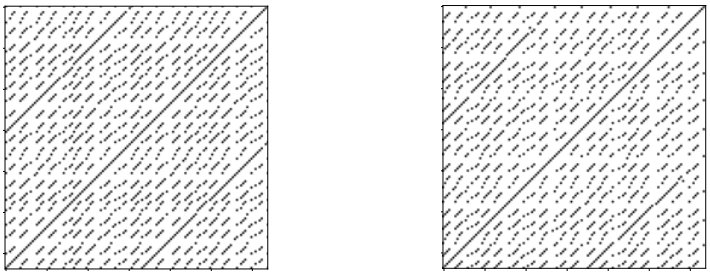
Figure 4.3: Excerpt of training data for datasets A and B

benign, DoS attacks, fuzzy attacks, and spoofing attacks look more distinct due to the large number of attacks sent. This is because the attack frequency value in dataset B is very low, making it easy for the IDS models to classify. As a result, we added a subtler dataset when comparing our proposed method with the Inception-ResNet-based method. When we prepare the recurrence plots, all of the arbitration IDs are considered because, in both datasets, all the arbitration IDs appeared in the CAN bus in a range of frequencies. There are cases of a CAN bus receiving a non-cyclic arbitration ID or a slight change in the sequence of arbitration IDs due to each ECU's clock skews. In such cases, deep learning should also learn this behavior of arbitration IDs.

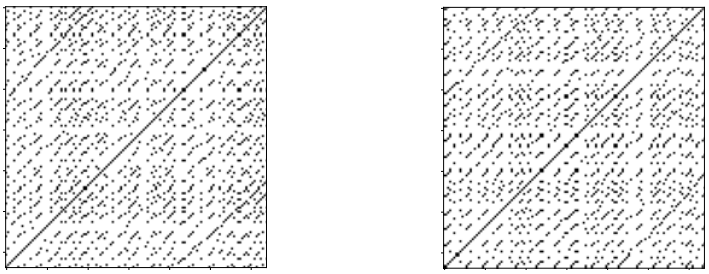
In the recurrence plots, we can see minor pattern variations that are captured in the images generated from the different attack types. The goal of this research would be to see if CNNs can learn this minor pattern variation. In doing so, it will be used to classify images into attack and non-attack types. We have experimented with binary and multi-class classification. In binary classification, the label for all the attack images is 1, and the label for the normal sequence is



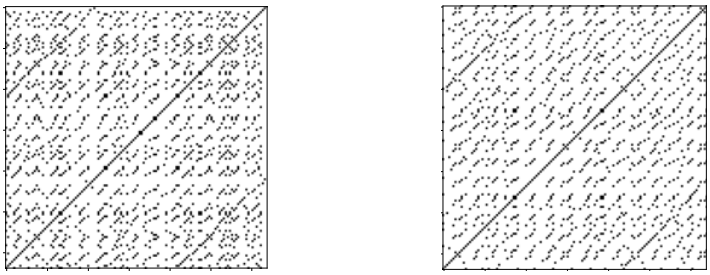
(a) Benign plots



(b) Drop attack plots



(c) Fuzzy attack plots



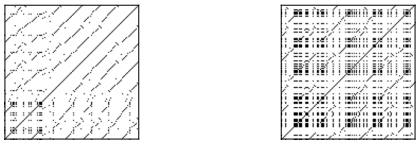
(d) Insertion attack plots

Figure 4.4: Recurrence plots of dataset A for all label types

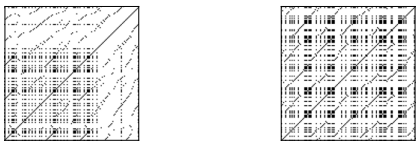
(a) Benign plots

(b) DoS attack plots

(c) Fuzzy attack plots



(d) Spoofing gear attack plots



(e) Spoofing RPM plots

Figure 4.5: Recurrence plots of dataset B for all label types

0. However, the multi-class classification has five class labels of 0, 1, 2, 3, or 4 in dataset B. The label 0 is for benign images, 1 for DoS attack images, 2 for fuzzy attack images, 3 for spoofing gear attack images, and 4 is for spoofing RPM attack images. Dataset A, only has 0, 1, 2, and 3 labels representing benign, drop, fuzzy, and insertion attacks. Both datasets have the same fully connected single node top layer with a sigmoid activation function. However, in the multi-class classification, the top layer has four nodes for dataset A and five nodes for dataset B with a Softmax activation function.

4.5.2 CAN attack simulations

We have collected a dataset to compare the proposed and the Inception-ResNet-based method. This is because the comparison was impossible in the public dataset as both methods result in near-perfect accuracy. Dataset A consists of benign data, drop attack data, fuzzy attack data, and insertion attack data. Except for the benign dataset, the simulated attacks are prepared by attacking a benign flow of packets. Attacking is performed in a period of random uniform time. Here are the details of how each attack is prepared:

- Drop attack: This attack simulates a remote attacker that drops packets from arriving at the connected ECU. The remote attacker can do so by compromising an ECU. This is done in our dataset by dropping a single packet in a period of selected length. A drop attack is similar to a DoS attack because packets are being dropped in both cases. A DoS attack is created by continuously flooding the CAN bus with a high-priority arbitration ID. The difference between the two is a drop attack does not need to send a new packet as a replacement for the dropped one, but a DoS attack sends packets in huge amounts for other packets to be dropped. We have used the time windows of 10ms, 50ms, 80ms, 100ms, and 150ms for all attack types. After the last packet is dropped, our attack system waits for a uniformly generated, random number between zero and the selected time window before making another attack. In our experiment, we have shown the performance of the proposed IDS on attack windows of 10ms, 50ms, 80ms, 100ms, and 150ms.

- Fuzzy attack: Attackers usually use this to learn how ECUs react to certain

packet types. A random CAN packet is inserted into the CAN bus to simulate this attack. This is different from the insertion attack because we have used a random arbitration ID that might not appear in the benign data of the arbitration IDs list.

- Insertion attack: As the name implies, an insertion is an attack done by inserting packets to the CAN bus. But unlike the fuzzy attack, insertion attack packets have genuine arbitration IDs and data frames.

Dataset B is publicly available at [83]. It contains five data types: DoS attack, fuzzy attack, "spoofing the gear" attack, "spoofing the revolution per minute (RPM) gauge" attack, and attack-free datasets. A DoS attack is prepared by injecting high-priority CAN messages in a short cycle of 0.3 milliseconds. Similarly, a fuzzy attack is prepared by injecting messages of spoofed random CAN ID and data values at a frequency of 0.5 milliseconds. The last Revolution Per Minute RPM/gear attack is prepared by injecting messages with CAN IDs related to RPM/gear to the CAN bus every millisecond.

The attack windows selected in the public dataset are very low compared to our dataset. If attacks occur more frequently, detection is easier, as there are more disturbances.

Figures 4.6 and 4.7 show the number of CAN frames in a one-second window for datasets A and B. Dataset A only has an average of 680 CAN frame in a second. Attacks are made in longer periods of 10ms, 50ms, 80ms, 100ms, and 150ms. For instance, in the 150ms period, there can only be as few as 6.6 attack frames, making it a lot more difficult than the public dataset. In dataset B, however, at least 1,953 frames are arriving at the CAN bus on average. When preparing the attack datasets, they have used time windows of 0.3ms, 0.5ms, and 1ms. This implies the one-second data may contain as much as 50% of attack packets.

In both datasets, a set of frames collected in a time window are labeled as an attack if there is at least one attack frame. It is labeled benign otherwise.

4.5.3 Network architecture

Our final target is secure in-vehicle network communication. We would like to detect as many attacks as possible, but implementing a very deep network that

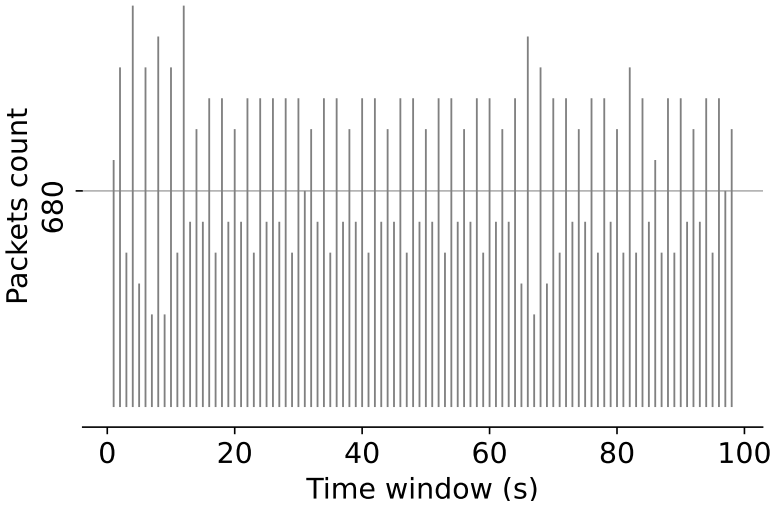


Figure 4.6: CAN frame count in a second, dataset A

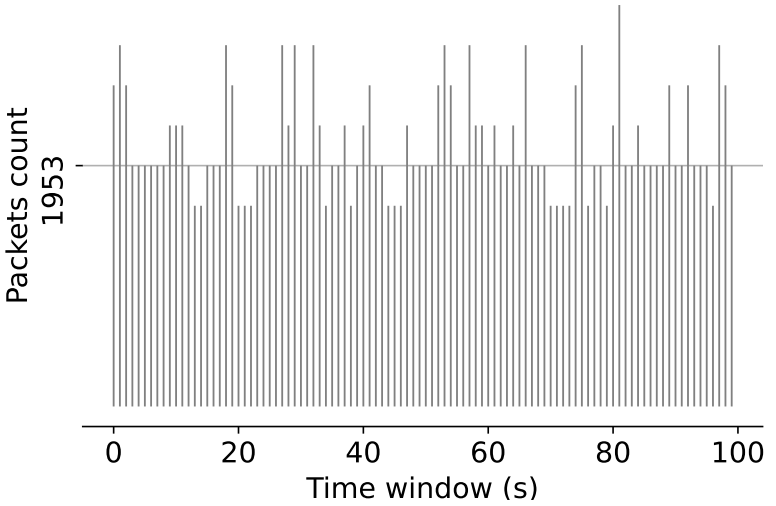


Figure 4.7: CAN frame count in a second, dataset B

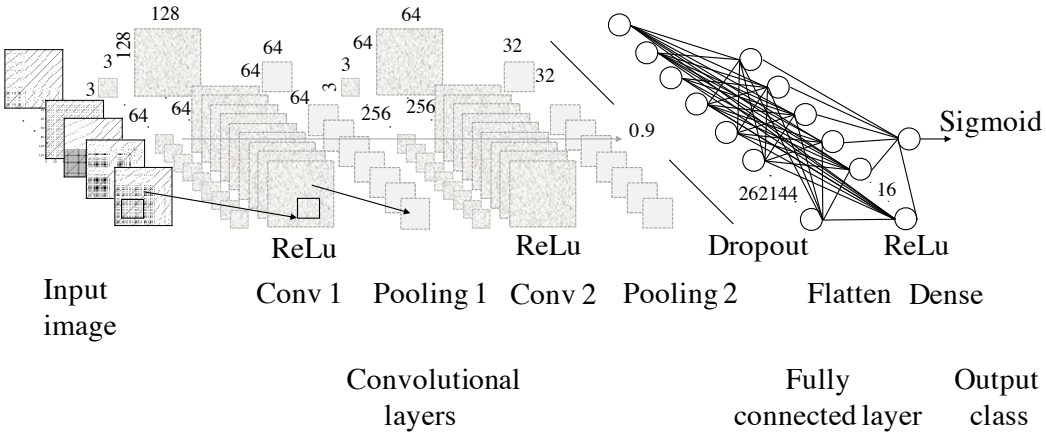


Figure 4.8: Binary anomaly classification

takes forever to make a prediction is not ideal. Therefore, we kept our network simple with only a few layers.

We have trained two classifiers: a binary classifier, and a multi-class classifier. The binary classifier classifies the images into two classes: benign and anomaly. A convolutional layer and a max-pooling layer are assembled in two stages. As shown in Figure 4.8, the first convolution has 64 filters with a kernel size of 3×3. For the output to have the same height/width dimension as the input, we have evenly padded the input image with zeros to the left/right and up/down. The activation function used in this layer is ReLu. The results from the first convolution are diminished with a max-pooling layer of a dimension of 2×2, no padding, and a stride of 2. The second convolutional layer has 256 filters, each with a kernel size of 3×3. Like the first layer, this layer also has a ReLU activation function and padding to keep the dimensions of the resulting matrix. Next to the convolutional layers, there is a dropout layer used for regularizing the network to fight overfitting. Since the output from the convolutional layers cannot be used directly, we flattened the resulting matrix to a one-dimensional vector. This one-dimensional vector is then fed to a dense layer with 16 nodes. In the top layer, there is one more feed-forward layer with only a single node and an activation function of the sigmoid. The sigmoid activation function gives a value between 0 and 1. If the resulting value from the layer is above 0.5, it will be classified as an anomaly image and otherwise benign.

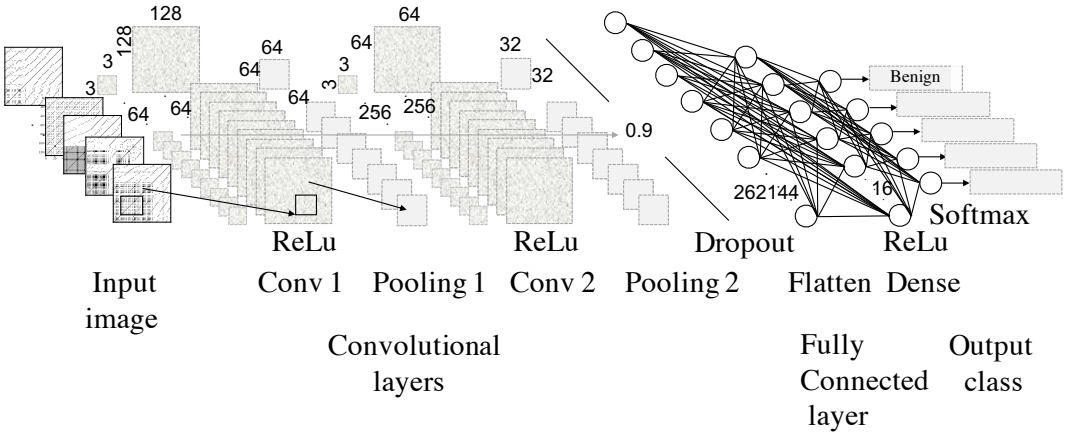


Figure 4.9: Multi-class anomaly classification

We have also created a multi-class classifier, Figure 4.9, to see if CNN can correctly classify the different types of attacks to their correct labels. It has a network architecture that is similar to the binary classifier, except for the top layer. Unlike the binary classifier's, the top dense layer has four or five nodes, depending on the type of dataset used. The resulting values from these nodes are finally fed to a Softmax activation function that gives probabilistic values of multi-class labels. The predicted class would be the one with the highest probabilistic value.

4.5.4 Hyperparameter search

In the current stage of deep learning, there is no easy way to set hyperparameters which often requires more manual work done manually or work done with computationally expensive hyperparameter search algorithms, like a grid search or a random search [84]. Without a good set of hyperparameters, the final model might not have a small training error and have a good generalization. To optimize the hyperparameters of our architecture, we have used an algorithm called Hyperband that speeds up the random search algorithm through adaptive resource allocation and early stopping [79].

Algorithm 1 is the Hyperband algorithm for searching hyperparameters. The algorithm requires two inputs: R, the maximum amount of resource that can be allocated to a single configuration, and η, an input that controls the proposition

Algorithm 1 Hyeprband Search

input: R,η (default $\eta = 3$)
initialization: $s_{max} = \log_\eta(R)$, $B = (s_{max} + 1)R$
for $s \in \{s_{max}, s_{max} - 1, \ldots, 0\}$ **do**
  $n = [\frac{B}{R}\frac{\eta^s}{(s+1)}]$, $r = R\eta^{-s}$
  $T = get\_hyperparameter\_configuration(n)$
  for $i \in \{0, \ldots, s\}$ **do**
    $n_i = [n\eta^{-i}]$
    $r_i = r\eta^i$
    $L = run\_then\_return\_val\_loss(t, r_i)$
    $T = top\_k(T, L, [ni/\eta])$
  end for
end for
return *Configuration with the smallest loss*

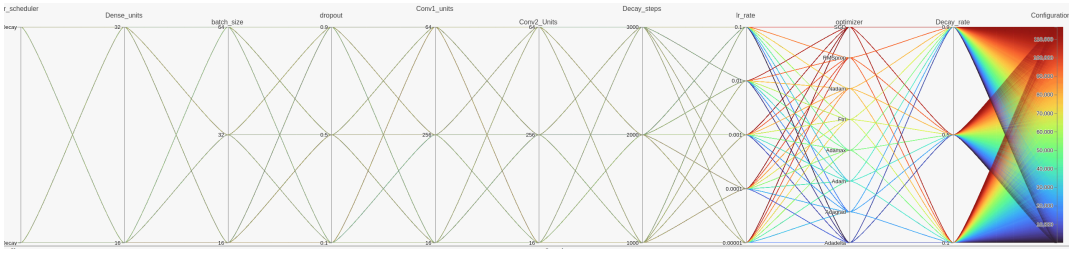


Figure 4.10: Combination of hyperparameters

of configurations discarded in each round of the inner for loop. Applying the algorithm requires the implementation of three methods. The first method, $get\_hyperparameter\_configuration(n)$, returns a set of n random identical and independently distributed (i.i.d) configurations from the total hyperparameter configuration space. The total number of configurations we used reached 38,880. These configurations are created by uniquely combining the hyperparameter, Figure 4.10, with values shown in Table 4.1.

The second method, $run\_then\_return\_val\_loss(t, r)$, takes a hyperparameter configuration t and a resource, r. The method trains the model for r epochs and returns the validation loss. This validation loss is then used in the $top\_k(configs, losses, k)$ method with the corresponding configurations. The

Table 4.1: hyperparameter values used for searching network architecture

Hyperparameter type	**List of values**
Convolutional Layer 1	16, 64, 256
Convolutional Layer 2	16, 64, 256
Dropout	0.1, 0.5, 0.9
Dense nodes	16, 32
optimizers	Adadelta, Adagrad, Adam Adamax, Adamax, Ftrl, Nadam RMSprop, SGD
Learning_rate	0.1, 0.01, 0.001, 0.0001 0.00001
Learning_schedule	Inverse time decay, Exponential decay
Decay_steps	1000, 2000, 3000
Decay_rate	0.1, 0.5, 0.9
Batch_size	16, 32, 64

$top\_k(configs, losses, k)$ returns top k best performing configurations, and running these configurations until a single configuration remains will do the work.

4.6 Experimental results and discussion

In this section, we present the tools and techniques we have used for the experiment. We show the model's intrusion detection performance in the two types of datasets. We have implemented the model in a controlled CAN to check its execution time when used in a real-world environment.

4.6.1 Dataset collection

We have collected dataset A from a real vehicle doing its day-to-day activities. The vehicle is a public passenger vehicle: a city bus. We have connected our CAN data-collection tools to the car's OBD-II port, which is found under the steering wheel of most present-day vehicles. We have connected our data-collection tools to the city bus for more than a week. In this range of days, we have collected millions of CAN traffic. We randomly took eight hours worth of CAN data for our experiment, which adds up to 16 million packets. One does not need to collect data from the entire lifetime of a car for training. There are only countable arbitration IDs, and learning the sequence of these arbitration IDs would only require a little training data.

After collecting the datasets, we have prepared the simulated attacks for dataset A. Since we have experimented with binary and multi-class classifications, we have prepared two different data types. Table 4.2 shows the binary classification's data sizes. In the binary classification, we have only two classes, benign and attack. For our model to be unbiased, the two classes need to be proportional. Therefore, we split the 16 million datasets into two. The first half was left intact as a benign dataset, but we used the second half to simulate the three types of attacks: drop attacks, fuzzy attacks, and insertion attacks. Then the whole dataset is combined and split into training, testing, and validation datasets in the proportion of 60%, 20%, and 20%.

Similarly, the multi-class classifier is prepared from the 16 million packets with the same proportion for all the data types. Table 4.3 shows the data types'

Table 4.2: Binary classification datasets of dataset A

Attack frequency (seconds)	**Data type**	**#CAN frames** (millions)	**Number of images**
0.01	Benign data	8	62,499
0.01	Drop attack	2.5	15,167
0.01	Fuzzy attack	2.5	23,896
0.01	Insertion attack	2.5	23,894
		Total	125,456

proportional split. Unlike the binary classifier, this has four classes, and all the class datasets need to have a proportional amount of training data. Therefore, the dataset is split into four classes, each containing 4 million packets.

4.6.2 Evaluation metrics

We used the area under the ROC curve plot (AUC) to compare the proposed and Inception-ResNet-based methods. We selected AUC because it reflects the overall ranking performance of our proposed and the Inception-ResNet-based method. ROC curve is a graph; the x-axis represents the false-positive rate, Equation 4.7, and the y-axis represents the true positive rate, Equation 4.6. In the equations, TP, FN, FP, and TN are acronyms for true positive, false negative, false positive, and true negative.

$$TruePositiveRate(TPR) = \frac{TP}{TP + FN} \tag{4.6}$$

$$FalsePositiveRate(FPR) = \frac{FP}{FP + TN} \tag{4.7}$$

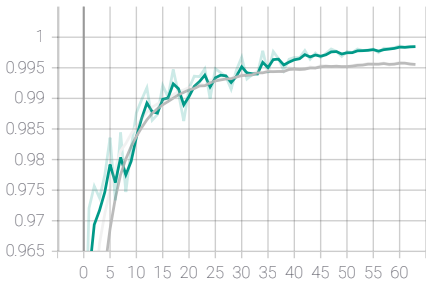
Table 4.3: Multi-class classification dataset of dataset A

Attack frequency	**Data type**	**#CAN frames** (millions)	**Number of images**
0.01	Benign data	4	21,201
0.01	Drop attack	4	18,838
0.01	Fuzzy attack	4	20,223
0.01	Insertion attack	4	20,219
		Total	80,481

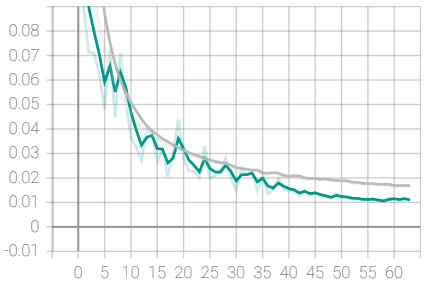
4.6.3 Model training

The best-performing model parameters are shown in Table 4.4. The selected architecture is trained on the Nadam optimizer with a 0.0001 initial learning rate, 0.1 decay rate, and a decay step value of the selected decay step multiplied by the ratio of training size to the batch size. The loss function we used is a binary cross-entropy for the binary classifier and a sparse categorical cross-entropy for the multi-class classifier.

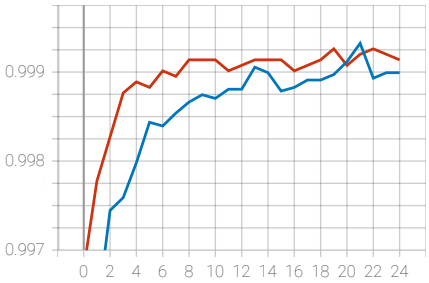
Using the hyperparameters shown in Table 4.4, the model is trained for 343 iterations with a constraint of early stopping when the model stops improving for more than ten epochs. The training metrics (loss and accuracy) from the training traces of both datasets are shown in Figure 4.11. We used validation data for hyperparameter tuning. Training/Validation accuracy and loss of dataset A are shown in Figures 4.11(a) and 4.11(b), respectively. Even though the total iteration selected is 343, the graphs do not show this. This happened due to the early stopping constraint we used during training. The model stops training when overfitting starts to show at around epoch 60 of dataset A. Similarly, Figures 4.11(c) and 4.11(d) show the training metrics of training and validation in dataset B. In this dataset, the early stopping constraint appears earlier at around epoch 24.



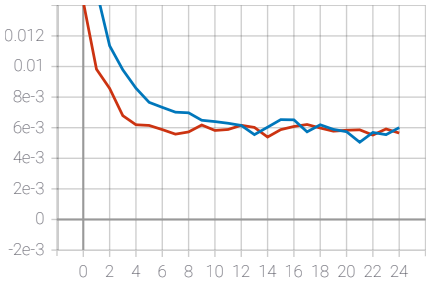
(a) Training accuracy (gray) and Validation accuracy (green) - Dataset A



(b) Training loss (gray) and Validation loss (green) - Dataset A



(c) Training accuracy (red) and Validation accuracy (blue) - Dataset B



(d) Training loss (red) and Validation loss (blue) - Dataset B

Figure 4.11: Training history plot of the binary classifier

Table 4.4: Hyperparameters of the architecture

Hyperparameter type	**List of values**
Convolutional Layer 1	64
Convolutional Layer 2	16
Dropout	0.9
Dense_nodes	32
optimizers	Nadam
Learning_rate	0.0001
Learning_schedule	Exponential decay
Decay_steps	1000
Decay_rate	0.1
Batch_size	32

4.6.4 Binary classification

The research aims to classify a sequence of arbitration IDs as benign or anomalous and take measures accordingly. In the binary classification type, the labeled dataset has only two labels, 0 for benign and 1 for an anomaly. On the top of the binary class classifier, we have used a sigmoid activation function. The sigmoid activation function takes an input of a multi-dimensional tensor and produces a value between 0 and 1. Depending on the output from this node, the classifier labels test data as benign if the output is below 0.5 and anomalous otherwise.

Tables 4.5 and 4.6 show the binary classification results of dataset A and dataset B. In datasets A, attack frequency refers to the time we wait to make a new attack after the last attack. In Table 4.5, the attack detection rate decreases as the attack window increases. When there are only a few attacks in the sequence, as low as only one, it gets difficult for the model to classify it to its correct label. This is because this kind of attack looks a lot like benign recurrent plots.

We have compared the results of our proposed method and the Inception-ResNet-based method. The ROC curves in Figure 4.12 show the ROC curve comparison results of the proposed and Inception-ResNet-based methods in dataset A. As

Table 4.5: Binary Classification results for dataset A

Attack frequency		**Predicted normal**	**Predicted attack**
0.01	True_normal	1.0	0.0
	True_attack	0.0	1.0
0.05	True_normal	1.0	0.0
	True_attack	0.004	0.996
0.08	True_normal	1.0	0.0
	True_attack	0.029	0.971
0.1	True_normal	1.0	0.0
	True_attack	0.045	0.955
0.15	True_normal	0.919	0.081
	True_attack	0.067	0.933

Table 4.6: Binary Classification results for dataset B

	Predicted normal	**Predicted attack**
True normal	1.0	0.0
True attack	0.002	0.998

Table 4.7: Parameter sizes comparison

	Parameter type	**Size**
Proposed	Trainable params	534,225
	Non-trainable params	0
	Total params	534,225
Inception-ResNet-based	Trainable params	1,691,490
	Non-trainable params	0
	Total params	1,691,490

we can see from the graphs, the Inception-ResNet-based works best when the attack frequency is high. Our model outperforms the Inception-ResNet-based significantly when there are only a few attacks in a window.

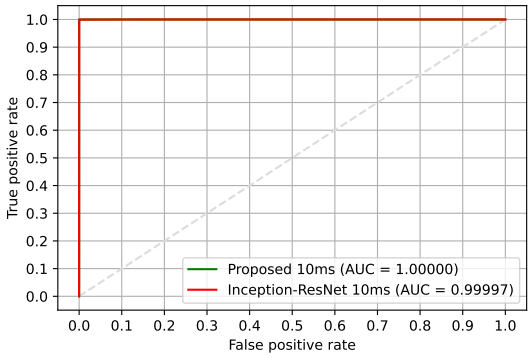
Similarly, we have compared the performance of the two methods in dataset B. The ROC curves in Figure 4.13 show how the proposed and Inception-ResNet-based performance on this dataset. We have implemented a recurrent plot with a window size of 128.

Models that have fewer parameters are preferred over large models for an intrusion detection system. This helps us in making inferences fast. Table 4.7 compares the parameter sizes of both Inception-ResNet-based and proposed methods. Figures 4.12 and 4.13 show that the proposed method outperforms the Inception-ResNet-based with yet fewer parameters, as shown in Table 4.7.

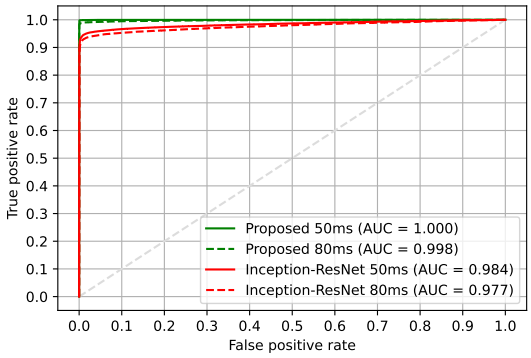
4.6.5 Multi-class classification

The multi-class classifier has the same architectural structure as the binary, except for the top layer. The top layer in the multi-class classifier has four nodes in training dataset A and five nodes for dataset B. The Softmax activation function in this layer gives the probability values for each class that sums to one. The one class with the highest probabilistic value will be selected during predictions. Figure 4.14 shows the multi-class classification results of both datasets.

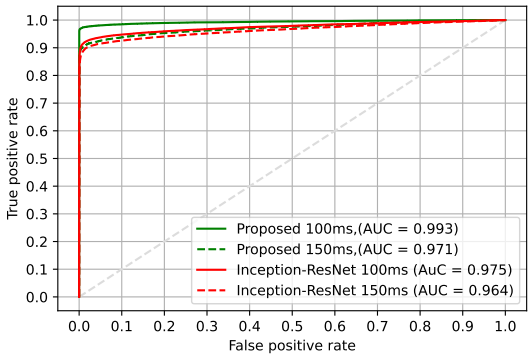
The multi-class classifier can be used when users would like to know about the properties of the attacks even though they fail in some cases. The multi-class classifier works better in dataset B than in dataset A. The attack types in dataset



(a) 10ms attack window



(b) 50ms and 80ms attack window



(c) 100ms and 150ms attack window

Figure 4.12: Proposed method ROC curve comparison with the Inception-ResNet-based Method. For all the tested attack windows

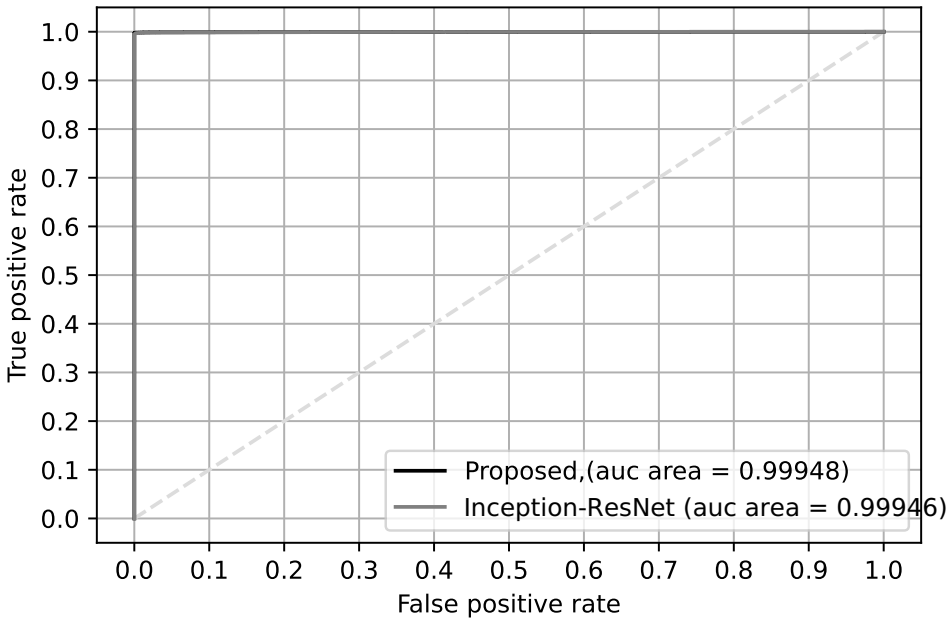


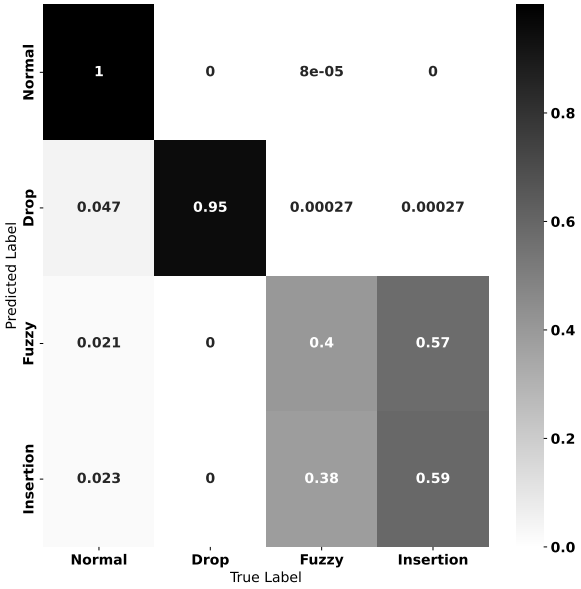
Figure 4.13: ROC of the proposed and Inception-ResNet-based method

A are distinct with no overlaps. But in dataset B, insertion and fuzzy attacks can sometimes present the same kind of images because the fuzzy attack is similar to the insertion attack when the attack ID selected is from the list of available arbitration IDs in the CAN bus. In such cases, the multi-class classifier fails to distinguish between insertion and fuzzy attacks.

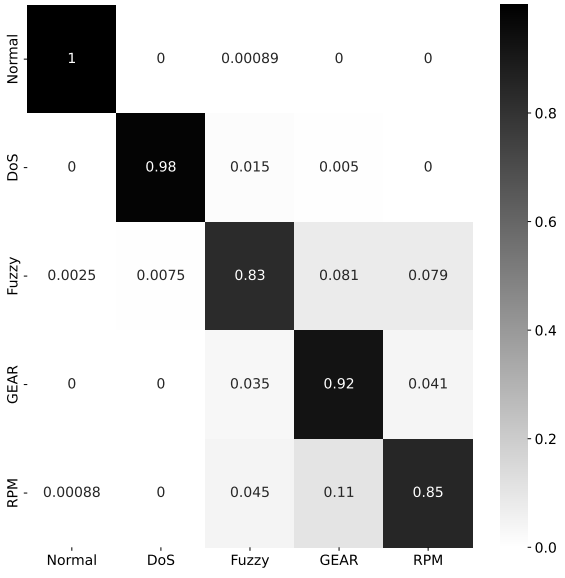
The hyperparameters selected for the binary classifiers are the same as those used for the multi-class classifiers. The main goal here is not to improve the accuracy but to show how the proposed method can be extended to a multi-class classifier. For better accuracy, it is recommended to do a hyperparameter search for this type as well.

4.6.6 Real-world IDS implementation

Deploying the proposed method to a vehicle for intrusion detection in the CAN bus requires an external host that will be connected to the CAN bus. This host needs to be capable of accessing the CAN bus data and making inferences



(a) Dataset A



(b) Dataset B

Figure 4.14: Confusion for multi-class classifier

Table 4.8: Jetson TX2's specification

Description	specification
GPU	256-core GPU
	max operating frequency: 1.12GHz
CPU	ARMv8 HMP CPU
	max operating frequency: 2.0GHz
Clocks	system clock: 38.4MHz
	sleep clock: 32.768MHz
Memory	type: 4ch x 32-bit LPDDR4
	max frequency: 1866MHz
	capacity: 8GB

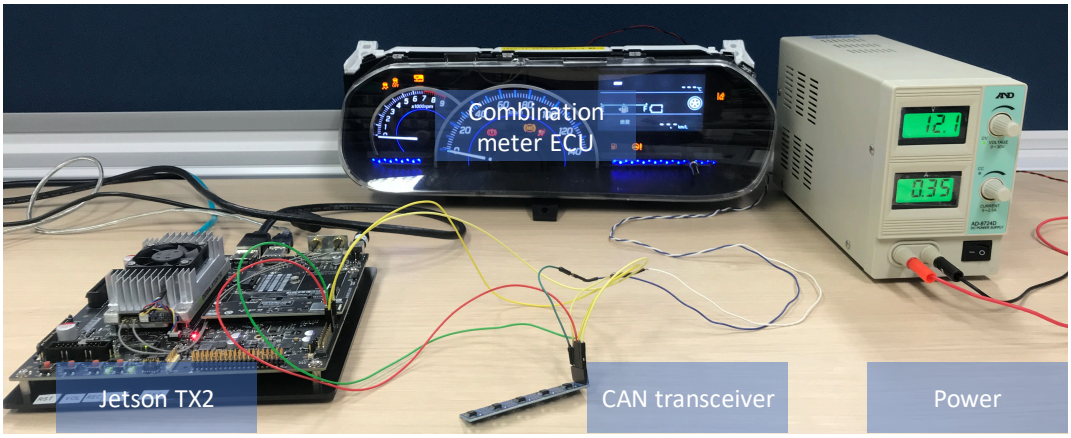


Figure 4.15: IDS real-world implementation

using the trained model. For this purpose, we have used NVIDIA's Jetson TX2. Table 4.8 shows the Jetson TX2's specification. The Jetson TX2 has 2 CAN controllers but no CAN transceiver. Therefore, an external CAN transceiver is required to access the CAN bus data. We have connected a CAN transceiver to J26 GPIO of the TX2, which later is connected to the CAN high and CAN low of the CAN network. Figure 4.15 shows the experimental IDS implementation. In the figure, the dashboard is extracted from a real vehicle that has an ECU capable of transmitting CAN frames to the Jetson TX2. The CAN transceiver is SN65HVD230 CAN Bus Transceiver Communication Module, enabling Jetson TX2 to access and interact with the CAN network.

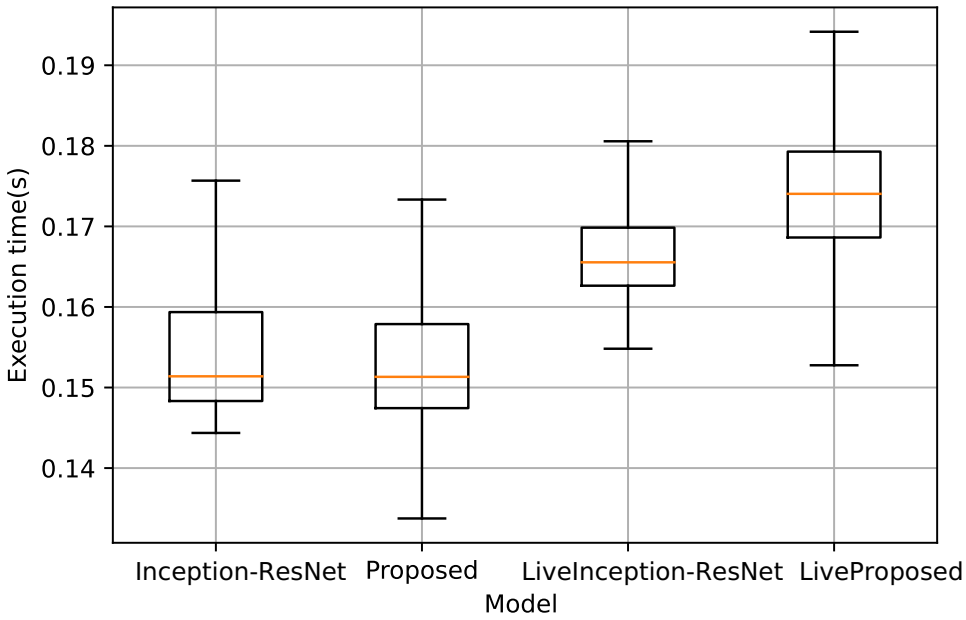


Figure 4.16: Model Execution Time

Our intrusion detection model makes two types of attack predictions: online and offline predictions. In the online prediction, the IDS system first collects packets from the real CAN network through the CAN transceiver. These collected packets are then preprocessed to make them suitable for the model input. The model then decides based on the model's inference results. In offline prediction, the model makes inferences for datasets preprocessed beforehand. The execution time for both cases is shown in Figure 4.16. The execution time of the live implementation is a bit slower than the offline implementation due to the preprocessing stage required for live implementation. The trained model will need an average of 117ms to make a single inference if Jetson TX2 [85] is used, but the current automotive controllers have lower processing power to execute deep learning models. Therefore, for this system to work, we need an external computational device like Jetson TX2 or similar devices.

If this system is going to be used as an IDS system in a real-world environment, what is trained on one vehicle can't be used as an IDS system in another. We need to create different models for each make and model of a vehicle. But a single

model in a vehicle can be used for any driving condition as long as most conditions are considered during training. Driving conditions might also not also affect the ID part of a CAN frame but affect the data part.

4.7 Conclusion

The proposed method trains a convolutional neural network model using images generated through recurrence plots. Recurrence plots can show temporal relations between arbitration IDs as most of the in-vehicle network arbitration IDs are periodic. Using these images, we have trained a CNN model that classifies in-vehicle network attacks into either binary or multi-class labels. We have compared the proposed with the Inception-ResNet-based method. The proposed method outperforms the Inception-ResNet-based method in performance with comparable execution time.

When we see neural networks in general, model retraining is required when the data used for training is significantly different from the data being collected for inference. Such situations can occur in our case when new arbitration IDs start to appear in the CAN bus due to software updates or similar situations. In such cases, we need to retrain the IDS model again from scratch.

Another shortcoming of the proposed method is that it can only detect attacks that disturb the CAN packet flow's normal sequence. Attacks like impersonation, which manipulate the CAN frame's data without affecting the arbitration ID, would be left undetected. In chapter 5 and chapter 6, we have extended the promising results found in this research to an IDS that can also detect such attacks.

5 Handling high-dimensional CAN bus data: MLIDS

The biggest disadvantage of training an IDS solely on arbitration IDs is that it will miss attacks that do not impact the sequence of arbitration IDs. We will be required to take a look into the data part of the CAN frames to detect such attacks. In this chapter, LSTM-based IDS is proposed that is capable of handling the high-dimensional nature of the CAN bus data.

5.1 Introduction

In this chapter, we propose an anomaly detection in the CAN bus using LSTM that is capable of handling the high-dimensional CAN bus data without the need for reverse-engineering of the CAN frames. Similar approaches are available, but all of them either require reverse-engineering of the CAN bus data or they don't handle the high-dimensional property of the CAN bus. In [73], the method proposed needed to implement a single architecture for all the available arbitration IDs. In [20], they handled the high-dimensional CAN bus data but their method requires reverse-engineering of the CAN bus data. Our research comes in handy in solving these two issues, avoidance of reverse-engineering and handling high-dimensional CAN bus data. We wanted to avoid the reverse-engineering part so as for the IDS system to work in all types of cars independent of the make and model. And the handling of the high-dimensional CAN bus data is required so that this IDS system could easily be optimized as part of other IDS systems. As it only provides a single anomaly signal, it can be used together with other IDS systems as part of one conditional statement.

5.2 Proposed method: handling high-dimensional CAN bus data

In this section we show how the high-dimensional structure of the CAN bus data is handled and the preparation of the anomaly signal. Our IDS system handles the high-dimensional structure of the CAN bus data by filtering each frame using its arbitration ID. Each filtered arbitration ID is fed to its corresponding LSTM models. Each of the models makes a subsequent frame prediction. The prediction errors from all of the LSTM models are later combined to create a single anomaly signal that can capture the attack state of the CAN bus in a certain time window.

5.2.1 Input data pre-processing

Dumping a CAN data gives row of packets sequenced by the time they arrived in the CAN bus. For this particular research, we have used the timing information, arbitration IDs and the frame values. The timing information is not used in the actual network training, it was only used for the sole purpose of keeping the sequence of the packets. Since packets from each of the arbitration IDs need to pass through a single LSTM architecture, the arbitration ID is used to distribute each of the packets to their respective architectures. After all the packets are filtered by their corresponding arbitration IDs, the packets which is in hexadecimal form is pre-processed so as for all the features to be binary.

5.2.2 MLIDS's network Architecture

In this section, we will describe the anomaly detection flow with the help of some notations. Let $\mathbf{F} = \{f_1, f_2, f_3, ..., f_n\}$ be the sequence of all, $n \in \mathbb{N}$, frames in order of their arrival time at the CAN bus. Before training, each frame $f \in \mathbf{F}$ is filtered according to its arbitration ID to create a data set that contains a sequence of frames that have identical arbitration IDs, $\mathbf{DS} = \{DS_{id_0}, DS_{id_1}, DS_{id_2}, ..., DS_{id_i}\}$. The model is trained in a time window of a second. In a second, some arbitration IDs, id_i, might appear more frequently than others, therefore the data set, **DS**, is a ragged tensor with unknown shapes. Each of the ragged tensors, $DS \in \mathbf{DS}$, observed in one second for each arbitration ID are fed to their corresponding

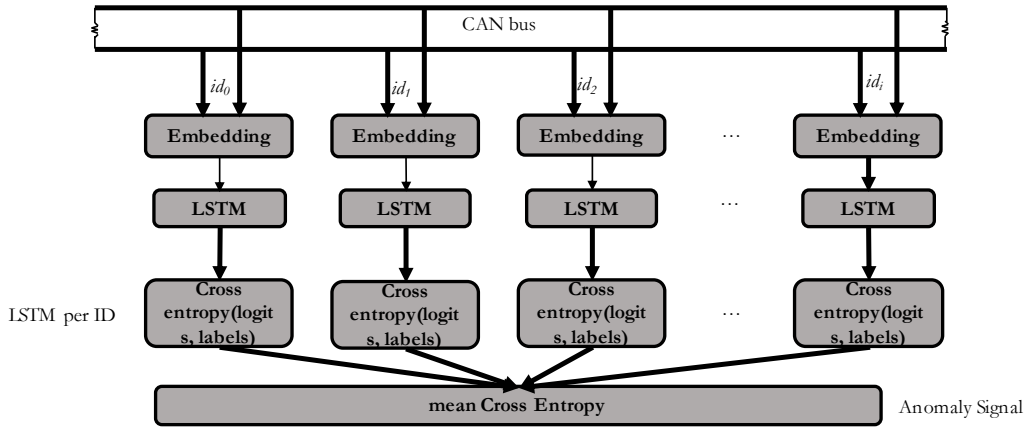


Figure 5.1: MLIDS system architecture

LSTM column as shown in Figure 5.1.

The main idea behind this architecture is to create a single anomaly signal that can be used to tell if there has been any kind of intrusion in the CAN bus. The detailed process that goes through one of the paths, let it be id_i, in the IDS system architecture is shown in Figure 5.2. In a second, all packets related to arbitration ID of id_i are collected and pre-processed as described in 5.2.1. During pre-processing, the collected packets are changed to input and output sequence that has a length of $k-1$, where k is the packets count in 1 second. The input contains all the packets from index 0 to index $k-1$ and the output contains elements from 1 to k. Packets 0 to index $k-1$ are then fed to the network to predict all the subsequent packets, $1'$ to k'. During backpropagation, we update all the weights connecting the layers using binary cross-entropy loss function. We trained the model until the binary loss between packets 1 to k and packets $1'$ to k' stops improving using early stopping and a hyper-parameter search algorithm introduced in 4.5.4.

Since the LSTM model we selected is stateful, the model can make predictions in any duration. If the selected time window is w seconds for $w \in \mathbb{R}$, the system makes predictions or anomaly searches in every w seconds. In a time window of w seconds, there might be an arbitrary number of packets, k, related to an arbitration ID of id_i. In this time window, the architecture predicts k number of subsequent packets. These predicted subsequent packets are then compared with packets which appeared in the CAN bus to calculate inner anomaly signal,

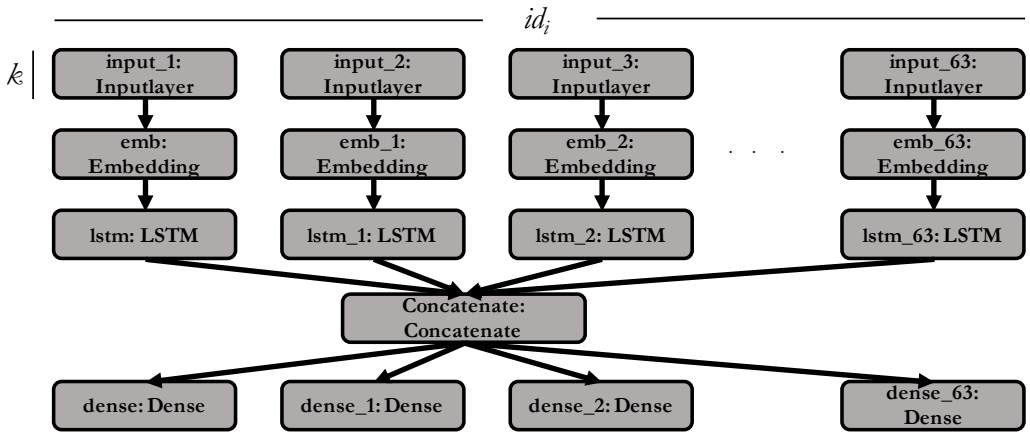


Figure 5.2: One path in the intrusion detection system

Pac_L, related to arbitration id_i. The CAN packet we used for training or testing is in binary format which means each value is a 0 or 1. As shown in Figure 5.2, each bit in a frame passes as an independent input to the architecture before it is concatenated in the lower layer. Inner anomaly signal is calculated using binary-cross entropy.

$$Pac\_L = -\frac{1}{64}\sum_{i=0}^{63}(y_i \log(p(y_i)) + (1 - y_i)\log(1 - p(y_i)) \tag{5.1}$$

The binary loss equation, Pac_L, gives the loss function in a single packet, but as we have returned all the sequences in the LSTM layer, we will have n number of loss values related to the total number of packets collected, k, in the time window of w. The arbitration ID level loss values are then calculated.

$$Arb\_L = \frac{1}{n}\sum_{j=0}^{n} Pac\_L_j \tag{5.2}$$

In a time window of w, each of the arbitration IDs, id_i, will each have a collective loss value. The final anomaly signal will be the mean results obtained from loss value of each arbitration ID. When calculating the final anomaly signal, frequency of the arbitration IDs is considered. If each of the arbitration IDs have a frequency as in $F_1, F_2, F_3, ..., F_i$ and arbitration ID level loss as in

$Arb\_L_1, Arb\_L_2, Arb\_L_3..., Arb\_L_i$ the final anomaly signal is calculated using 5.3.

$$Cumulative\_Loss = \frac{\sum_{y=1}^{i}(F_y * Arb\_L_y)}{\sum_{y=1}^{i} F_y} \tag{5.3}$$

5.2.3 Evaluation Metrics

We used recall, precision, and F_1 score to show the performance results of our proposed method. These metrics are commonly used in binary classification problems. Equations 5.4, 5.5, and 5.6 are used to calculate the metrics in order of recall, precision and F_1 score. In the equations TP is for "true positive", FN is for false negative and FP is for false positive.

$$Recall = \frac{TP}{TP + FN} \tag{5.4}$$

$$Precision = \frac{TP}{TP + FP} \tag{5.5}$$

$$F_1 score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{5.6}$$

5.2.4 Hyperparameter search

To tune the neural network's hyper parameters we used hparams algorithms from tensorflow [86]. Our parameter search goes through three phases. In the first phase, we run 108 configurations created with the combination of parameter values shown in Table 5.1. With a condition of early stopping callback that checks if the validation loss is improving in less than 15 epochs, all the configurations are trained for 200 epochs. The result from the first run of the hyperparameters is shown in the Figure 5.3.

In the second step of the hyperparameter search, 9 best results are selected for further run. These configurations are run for 200 more epochs. The configuration

Table 5.1: hyperparameter values used for searching network architecture

Hyperparameter type	Values
Embedding_dim	[4, 8 , 16]
RNN_layers	[1]
RNN_units	[32, 64, 128]
Learning_rate	[0.001, 0.01, 0.1]
Learning_schedule	[Inverse time delay, Exponential decay]
Optimizer	[SGD, RMSprop]

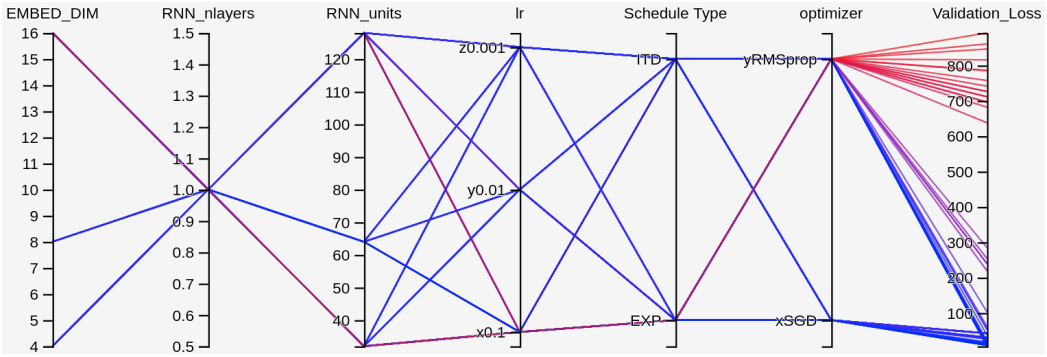


Figure 5.3: Hyperparameter search results

with the best result is then tuned again manually to see if there is a room for improvement. In the final step a single best hyperparameter is used to train and test the IDS system. The details of the selected hyperparameter values are described in 5.3.3.

5.3 Experimental Results and Discussion

5.3.1 Data set collection

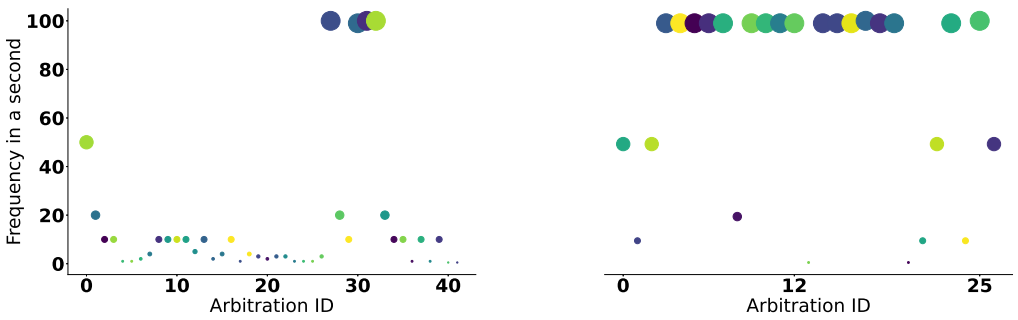
We collected data from a real vehicle, prv_data, for about 14 hours that count to a total of 36 million packets. We split the data set into a ratio of 70%, 15% and 15% for training, validation and testing respectively. Since CAN bus data is proprietary, we couldn't release our dataset for researchers to use. Instead we experimented MLIDS with a public CAN bus data from [19], pub_data. This dataset contains normal data (attack free), targeted attack (gear and RPM), DoS and Fuzzy attacks. To train the MLIDS we split the normal data to the same proportion as prv_data.

For both of the datasets, we tested the performance of MLIDS with a time window of 0.5, 1 and 2 seconds. For an arbitration ID to be used for training, it has to appear in the CAN bus in the minimum time window. As shown in Figure 5.4 , some of the IDs appear in the CAN bus less frequently in the time window of 0.5 seconds. Due to this we considered the top 21 arbitration IDs in both of the test data. In the prv_data, 21 arbitration IDs were considered because these arbitration IDs cover 94% of all the test car's CAN bus data. The top 21 arbitration IDs in the pub_data also cover more than 97% of the data.

5.3.2 simulated attacks

To evaluate the performance of our IDS system, we have fabricated three types of attacks in the the prv_data.

InsertionAttack - In every 0.01 seconds, a randomly selected genuine frame is inserted in a sequence. The duration of the attack depends on detection window selected during testing (0.5, 1, or 2 seconds). If the selected window is 0.5 seconds, the insertion attack is made in every 0.5 seconds.



(a) prv_data (b) pub_data

Figure 5.4: Average frequency of all available arbitration IDs in a second

DropAttack - We drop a single packet in every 0.01 seconds. This attack also continues for a duration of time selected during testing.

FuzzyAttack - This type of attack is often used to learn how ECUs react to injected frames. We fabricated this attack by randomly preparing a 64-bit frame. The arbitration ID for this frame is also randomly selected from the available arbitration IDs in the test car. Same as the other two attacks, fuzzy attacks period depends on the selected window.

But for the case of pub_data, all the data is used as it is except for the DoS attack. In this dataset arbitration ID '0000' is sent in high frequency to create a DoS attack in the CAN bus. But, this particular arbitration ID is not in the IDS. Therefore, we replaced this arbitration ID in the DoS test data with an ID that has a highest priority, arbitration ID '0002'.

5.3.3 Network training results

The network architecture is carefully selected using a hyperparameter search algorithm called Hparams. The best architecture has an embedding layer, LSTM layer and a dense layer, that is used to predict the forthcoming frame bits. An 8 bytes long packet has 64 bits with each being 0 or 1. The architecture predicts the probability of each of the 64 bits. These values are considered categorical with two classes, 0 and 1. To handle these categorical values, we used embedding in the first layer with 16 units. The output from this layer is fed to a single

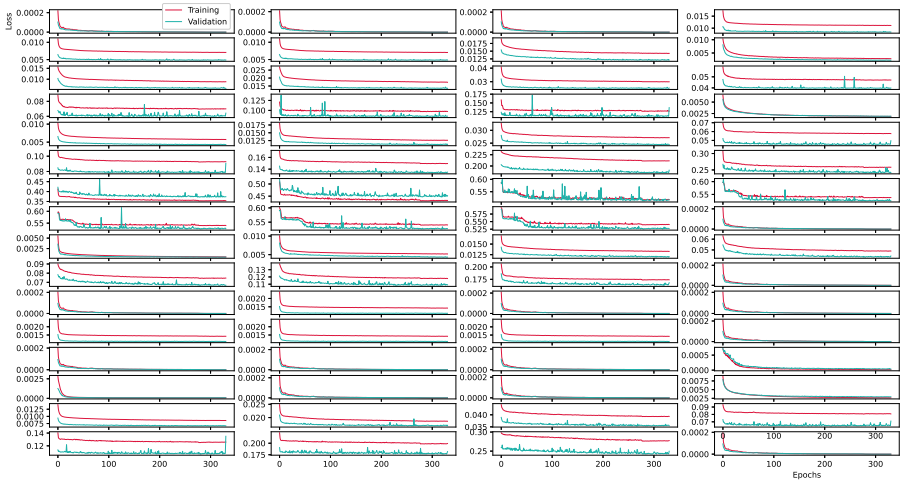


Figure 5.5: The first arbitration ID's all 64 bits training and validation loss values

LSTM layer with 32 units. During searching for hyperparameter, only a single LSTM layer is considered due to its expensive computations that is not suitable for a slow-performing network like CAN bus. At the top of the architecture, we have a dense layer that predicts the probability of the frame being all 1's with an activation function of sigmoid. The loss function used is a binary cross-entropy between labels from the training data and output from the dense layer. The loss function combined with an optimizer of Stochastic Gradient Descent (SGD) that has a momentum of 0.8 and a clip value of 1.0 is used to compile the model. We have also tuned the learning rate upon which SGD is learnt. The learning rate is scheduled for the SGD using Inverse Time Decay.

Figure 5.5 shows the training and validation loss values of the first arbitration ID considered in prv_data. As it can be seen in the figure some of the bits in the packets are not fitting well while the others are perfectly fit. This is because it is impossible to find a collection of hyperparameters that works best for all the bits. When tuning the architecture, the average loss of all the bits is checked for early stopping.

The trained model makes a prediction in a specific time window. The anomaly signal is calculated from a loss between the predicted values and true values.

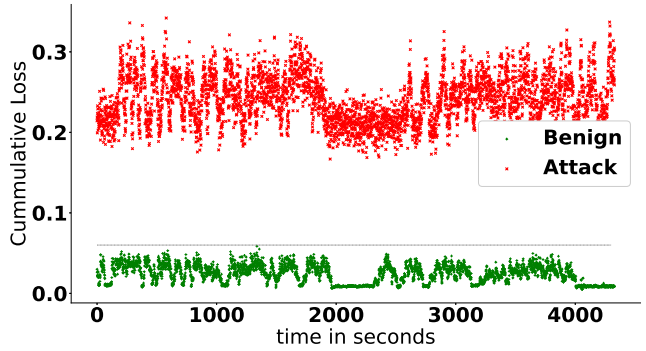
Table 5.2: Prv_data detection results for all types of attacks and window of duration

Duration	Metrics	Insertion	Drop	Fuzzy
0.5	recall	1.0	0.96	1.0
	precision	1.0	0.52	1.0
	F1_score	1.0	0.68	1.0
1	recall	1.0	1.0	1.0
	precision	1.0	1.0	1.0
	F1_score	1.0	1.0	1.0
2	recall	1.0	1.0	1.0
	precision	1.0	1.0	1.0
	F1_score	1.0	1.0	1.0

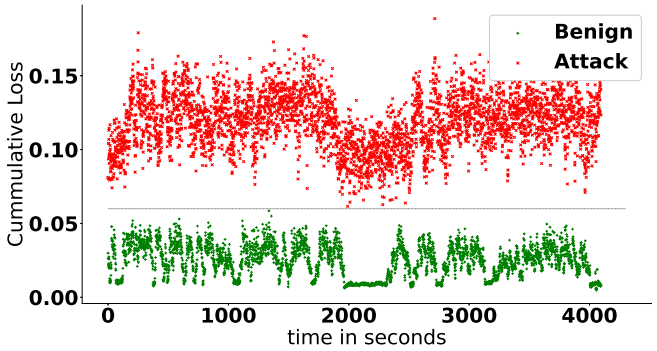
Figure 5.6 shows, scattered plot of the anomaly signal calculated for insertion, drop and fuzzy attacks in 1 second.

After the results are scattered as the result figures, a threshold is manually selected that gives maximum performance for all types of attacks and window sizes. Table 5.2 and Table 5.3 show precision, recall and F_1 score results of all the cases. As we can see from the result tables, the system effectively detects all types of attacks except for drop attack in prv_data. This is because some windows have a lower frequency of frames and dropping a frame in between will not have a big of an effect on the anomaly signal calculations.

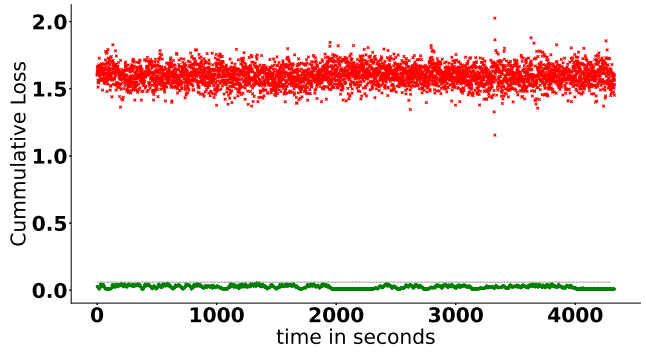
The IDS system can effectively incorporate any size of CAN frame. For most present-day cars which use the CAN network, all that is needed to use this implementation is extract the bits in a CAN frame and train a model with the architecture presented. The training and threshold selection will be done offline as part of the IDS system. The final IDS system would be a trained model that will be placed anywhere in a car where there is a full access to the CAN bus. Depending on the status of intrusions, the IDS system can be used to notify the driver about attacks. In times where this system comes short, it can be used in collaboration with other types of IDS systems. To make this simple, we have found a way to prepare a single anomaly signal that can be easily used with a conditional statement.



(a) Insertion Attack



(b) Drop Attack



(c) Fuzzy attack

Figure 5.6: Scattered plot of results in the prv_data in a window of 1 second

Table 5.3: Pub_data detection results for all types of attacks and window of duration

Duration	Metrics	DoS	Fuzzy	RPM	Gear
0.5	recall	1.0	0.999	1.0	1.0
	precision	0.999	0.999	1.0	1.0
	F1_score	0.999	0.999	1.0	1.0
1	recall	1.0	1.0	1.0	1.0
	precision	1.0	1.0	1.0	1.0
	F1_score	1.0	1.0	1.0	1.0
2	recall	1.0	1.0	1.0	1.0
	precision	1.0	1.0	1.0	1.0
	F1_score	1.0	1.0	1.0	1.0

5.3.4 MLIDS execution time

The execution time it takes for the trained model to make a single packet prediction is shown in Figure 5.7. The IDS system is implemented in Ubuntu 18.04 OS, Intel Xeon E5-1620 CPU and GM200 (GeForce GTX TITAN X)3072 CUDA cores GPU that has a clock speed 33MHz and a RAM size of 16GB. The prediction time is almost similar in all time windows.

5.4 Conclusion

For a secure CAN communication protocol, in this work we have presented an intrusion detection system using LSTM. The novelty of this work is the ability to handle multidimensional data without the need for reverse-engineering of the training data. Our research managed to extract a single anomaly signal for a window of seconds. In a real-time communication a delay of 1 second in a prediction might have a disastrous effect. To improve the prediction delay, we will continue researching on placing the trained model in a cloud and see if this delay could be improved.

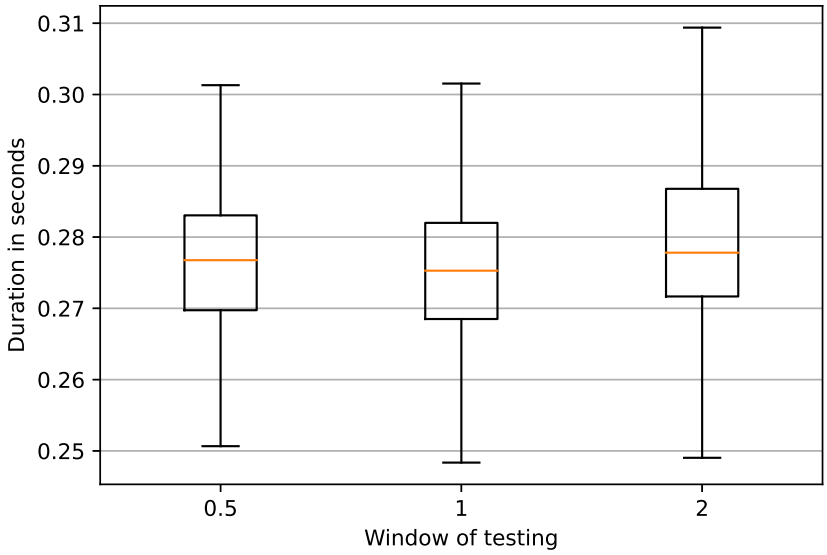


Figure 5.7: Execution time of the IDS system in making predictions

6 Application of image segmentation in the CAN bus intrusion detection: U-CAN

This chapter also used CAN data portion instead of arbitration ID. This way, it can be possible to detect attacks that keep ID sequences intact. As an improvement to MLIDS, this chapter discusses the use CNN on the data portion of the CAN bus data for intrusion detection.

6.1 Introduction

In this chapter a CNN-based intrusion detection for CAN bus is proposed. Since the proposed work has similar network architecture as U-NET [87], it will be referenced U-CAN. U-CAN takes the advancements in CNN's image segmentation to in-vehicle network intrusion detection. Previous research works have used CNNs to detect CAN intrusions, but the vast majority of studies have dealt with either raw CAN data or reverse-engineered CAN data. Unlike previous studies, U-CAN can be used for both types of data. U-CAN uses a hamming distance (HAMD) distribution of CAN frame bits to deal with raw CAN frames that are used to train a model. The trained model can be deployed to listen in the OBD II port of vehicles for intrusion detection. We have also added a rule-based system that checks the HAMD of counter bits in a CAN frame. The rule-based system flags any sequence that deviates from a predefined hamming value. For reverse-engineered CAN frames, we applied a saliency detection algorithm before feeding the data to U-CAN. This helps us to easily segment attack windows of the CAN signal sequences.

The following is a summary of the research's significant contributions:

- U-CAN is the first one-dimensional (1D) segmentation method applied to in-vehicle networks for intrusion detection, which can be applied to both raw and reverse-engineered CAN frames.
- It is the first work that studies the pattern of HAMD in CAN frames for in-vehicle network intrusion detection.
- U-CAN is tested in a publicly available dataset of both raw and reverse-engineered CAN frames resulting in promising F_1 *Score*s.

6.2 CAN bus data and adversary model

Despite the fact that CAN is standardized, vehicle manufacturers determine the meaning of each CAN frame. Each vehicle's CAN frames may also differ depending on the brand and model. One needs to reverse engineer these proprietary frames to understand how vehicles react to certain CAN frames. U-CAN can be used in both raw and reverse-engineered CAN frames. In the case of the raw CAN frames, a reverse engineering algorithm is used from [88]. But in the second scenario, we deal with the physical value of signals that have been reverse-engineered to a human-readable format and is available at [20].

Two datasets are considered in the experiment, which we will be referring to as the RAW dataset and the PHY dataset. The RAW dataset is publicly available at [17]. The dataset contains five data types: DoS attack, fuzzy attack, "spoofing the gear" attack, "spoofing the revolution per minute (RPM) gauge" attack, and attack-free datasets. As per the work by [88], a CAN frame can contain continuous values (e.g., speed), pseudo-random values (e.g., checksum), enumeration signals (e.g., door open status), and cyclic signals (e.g., counters and clocks). To understand which of the CAN frame bits represent the different CAN signals, we need to reverse engineer the dataset using HAMD distribution. This distribution helps us in identify the bits in a frame that are used as counter bits. HAMD is computed by dividing the number of bit flips by the total frames that have arrived on the bus in a duration of a selected time window. Figure 6.1, shows the HAMD of the RAW dataset for arbitration ID 0130. The top three bits in the

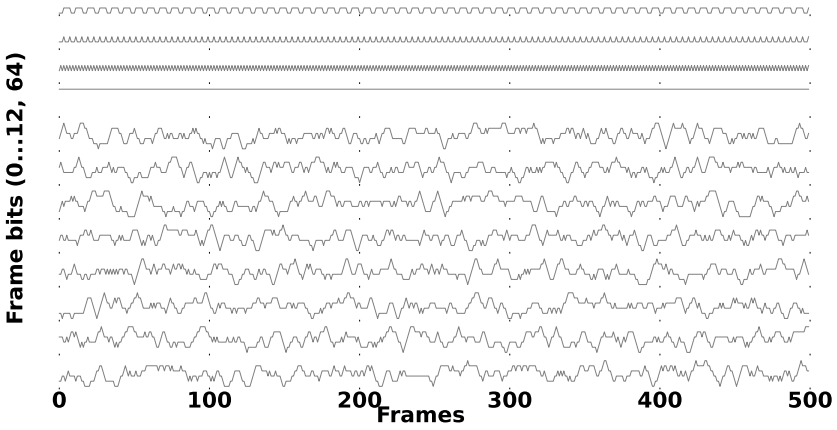


Figure 6.1: Hamming distribution of arbitration ID 0130 of RAW dataset.

figure represent the counter bits in this particular arbitration ID. We will follow along with these kinds of bits for all arbitration IDs to detect attacks in the CAN bus.

The second dataset, the PHY dataset, is publicly available at [20]. The dataset contains benign and attack sequences of reverse-engineered and preprocessed CAN signals. It contains 10 arbitration IDs, each with different sizes of signals to represent the information carried in a single CAN frame. There are a total of 20 signals with all the arbitration IDs. In our work, we will extract all the signals and train a single model for each arbitration ID. The dataset contains a train and testing dataset. Since it got released with encoder/decoder-based IDS, the training dataset contains only benign data. The only dataset with attack data is the test dataset. In our case, we will be training a classifier network, and hence, we would need to simulate similar types of attacks in the training data.

We first need to understand how the test data is prepared in the dataset released. The test data for the plateau attack is prepared by overwriting true signals with a constant value over time. Figure 6.2 shows the plateau attack of arbitration ID 10, which has four signals and a label to show which of the signal values are manipulated. As shown in the figure, if we want to repeat this attack on our training data, all we need to do is select an attack window and replace the values

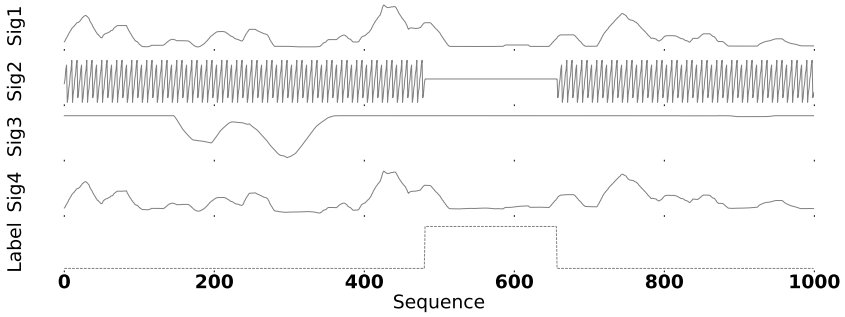


Figure 6.2: Plateau attack of Arbitration ID 10 which has four signals

in the window with a randomly selected constant signal value. In Figure 6.2, it is the second signal from the top that was attacked to simulate the plateau attack.

Similar to the test data, we have simulated plateau attacks on the training data. Figure 6.3 shows a box plot of the time difference between two frames of the same arbitration ID. In each attack window, the test data selects a single arbitration ID signal to be taken over by a single constant value. This attack continues for a random time window of between 4,169 ms and 8,332 ms. We have followed this exact approach to create plateau attacks in our training data. Attacks in our simulation are made for a random uniform time between 4,169 ms and 8,332 ms. Once the attack window time has elapsed, we leave the frames intact for a random uniform time between 16,000 ms and 24,000 ms before making the next attack. Our neural network model will be trained on this simulated training data and tested on the real attack frames.

6.3 U-CAN: Securing the CAN network through convolutional neural networks

This section begins with a quick overview of deep learning frameworks that are used in the IDS model. These frameworks include CNNs and encoder/decoder models. In the later sections, we will describe in detail the preprocessing stages we followed to finally train the U-CAN model.

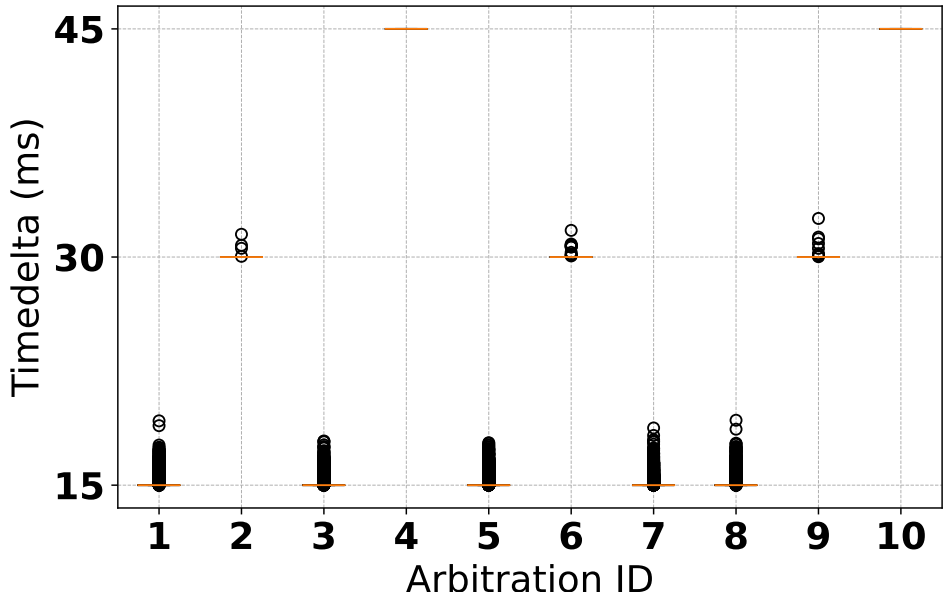


Figure 6.3: Boxplot of the time difference between consecutive frames of the same ID

6.3.1 Building blocks of U-CAN

To segment timestamps of attack windows, U-CAN employs an encoder/decoder technique. The work is an extension of the U-NET image segmentation system, which was the first to be used in the segmentation of medical images [87]. The model consists of an encoder path and a decoder path that is built on blocks of CNNs. The network architecture of the U-CAN model is similar to the one used in the sleep staging paper [89]; however, unlike the U-NET model, which was applied to 2D images, our work, similar to U-Sleep, is applied to a 1D dataset. Figure 6.4 shows the U-CAN network architecture. U-CAN, which is similar to U-Sleep in that it consists of an encoder, a decoder, and a segment classifier module.

The encoder module takes an input of a single arbitration ID signals. We have used a fixed size of single arbitration ID signals to train the model which we will refer to as sequence length, (S). During testing, inferring is done for a number of windows instead of a single window. The number of windows selected is referred to as period (P). Therefore, the total signal train size would be $S \times P$. The encoder then takes this input and encodes it through 12 blocks. Each block

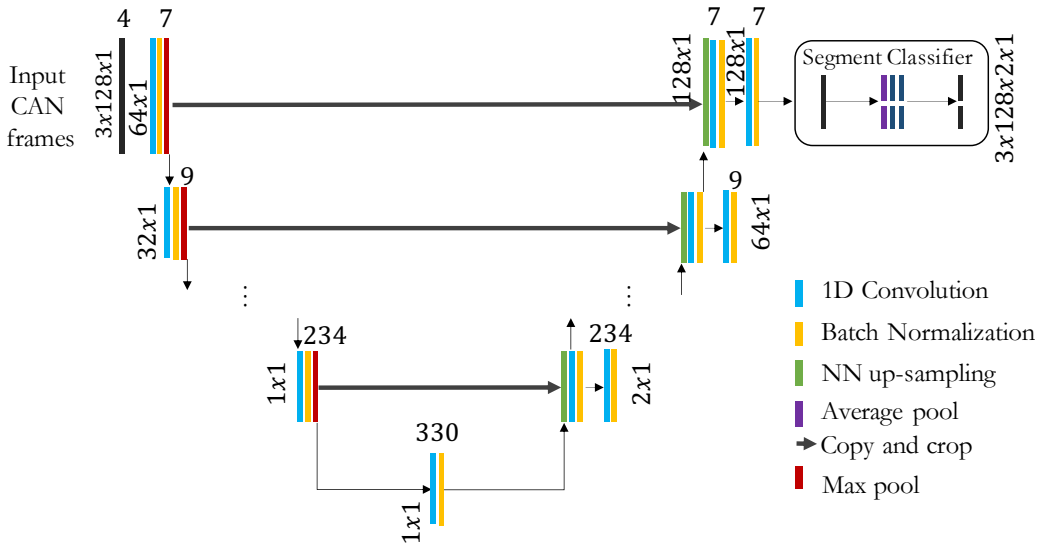


Figure 6.4: Model Architecture

consists of a convolutional layer, batch normalization, and max-pooling layers. The convolutional layer has a kernel size of 9 with no kernel dilation, a stride of 1, and an activation function called ELU. Unlike these parameters, the filter of the convolutional layer has an initial filter of 5 and decreases by a ratio through all the blocks of the encoder. The parameters for batch normalization are left with the default values of the Tensorflow implementation. The last layer in the block is a max-pooling layer with a kernel size of 2 and a stride of 2. The main success behind the U-NET model is the use of layer outputs in an encoder as an input to the decoder. Similar to the U-NET model, U-CAN also collects residual connections from the batch normalization of each block in the encoder, which can later be reshaped and concatenated with the decoder's batch normalization output.

The decoder/up-sampler part also consists of 12 blocks, with each containing an up-sampling layer, a convolutional layer, a batch normalization layer, a concatenating layer, and a second batch normalization layer. The up-sampling layer has a kernel size of 2 for doubling the length of the feature maps. The convolutional layer has a kernel size of 2 and a stride of 1 with ELU activation. The residual connections from the encoder are cropped and concatenated with the outputs of the batch normalization layer, which is next to the convolutional layer. The

Table 6.1: List of hyperparameters

Hyperparameter	Search parameters
Batch size	16, 32, 64
Learning rate	0.1, 0.01,0.001, 0.0001,0.00001
Learning decay	0.1, 0.5 0.9
Learning steps	10000, 20000, 30000
Scheduler	Exponential Decay, Inverse Time Decay
Optimizer	Adadelta, Adagrad, Adam, Adamax, Ftrl, Nadam, RMSprop, SGD

decoder then finishes with one more batch normalization layer, which takes the concatenated output from the previous layer. The last module is the segment classifier, which is used to classify each window of signals to its correct class.

The last module, the segment classifier, takes outputs from the decoder to attack prediction to the same size as the sequence length. The selected sequence length is 128, which is also the parameter for the average pooling layer at this stage. These results are then fed to two point-wise convolution operations (with a kernel width of 1 and stride of 1) that have ELU activation and Softmax activation functions, respectively. The Softmax classifier at the last convolutional layer gives a probabilistic prediction for each of the available classes. The network architecture and parameters selected are similar to U-Sleep [89]. We have only tuned the hyperparameters found during training that are outside of the network layers. These hyperparameters include: the batch size, learning rate, learning decays, learning steps, schedulers, and optimizer. Hyperband [79], an algorithm that takes advantage of random search, is used to search through the hyperparameters listed in Table 6.1.

6.3.2 Data preprocessing

As introduced earlier, this IDS is used in two datasets: RAW dataset and PHY dataset. Not much preprocessing is done in the RAW dataset, except for applying HAMD to the dataset. HAMD helps us in identifying counter-bits in CAN frames. These counter bits are first studied carefully to learn a rule-based IDS and U-CAN,

Table 6.2: Sample dataset

Label	Time	ID	signal1	signal2	signal3	signal4
0	2088.41...	id5	0.0	0.96	-	-
0	2089.55...	id8	0.25	-	-	-
0	2090.88...	id3	0.2	1.0	-	-
0	2091.65...	id7	0.06	0.0	-	-
0	2100.36...	id1	0.46	0.11	0.95	0.17

which is an extension of the rule-based IDS. The U-CAN used in this dataset is not much more than a segmentation model that segments time windows with disturbed HAMD of the counter bits.

The second dataset, PHY dataset, includes multi-dimensional data. Each arbitration ID carries a variable number of signals. We have trained a single U-CAN model for each arbitration ID after concatenating zeros to make the signal size four as the maximum number of signals carried in one arbitration ID is four. The sample dataset for the training data is shown in Table 6.2.

After the attacks were simulated as explained in the previous sections, we extracted frames of each arbitration ID to train the U-CAN architecture. However, training the model with the available format of CAN signals did not yield us good results. We further preprocessed the signal values by applying the spectral residual model to all the training and testing data. This changes the problem we are trying to solve from detection of attack windows to saliency detection. The spectral residual model was first applied in image saliency detection by [90]. Its application is also extended in time series anomaly detection by [91]. Similarly, we have applied a spectral residual algorithm to each signal value of an arbitration ID.

Given an arbitration ID a, with four signals, $a = [s_1, s_2, s_3, s_4]$. The sequence of frames for this particular arbitration ID would be $A = [a_0, a_1, ..., a_i]$, $i \in \mathbb{N}$ representing the total CAN frames of ID a. For each column of A, we have used a spectral residual algorithm that is mathematically represented in equations 6.1, 6.2, 6.3, 6.4, 6.5, and 6.6 applied in the respective order of their appearance. In the equations, $\mathfrak{F}$ and $\mathfrak{F}^{-1}$ denote Fourier Transform and Inverse Fourier Transform, respectively. For each column of signals, $A[:, s_i]$, in A, we

apply the spectral residual algorithm, which starts with applying the Amplitude spectrum, $A(f)$, of the Fourier transformed input. $P(f)$ is the phase spectrum; $L(f)$ is the log transformation of $A(f)$; $AL(f)$ is the average spectrum of $L(f)$ after convolving it with $h_q(f)$ that is a square matrix of one's with shape $q \times q$. When applying the log transformation, we used an ϵ value 10^{-8} that overwrites the resulting value to 0 for items less than the ϵ. $R(f)$ is a spectral residual calculated from $L(f)$ and $AL(f)$. In this equation, we need moving average values of the sequence. The moving average size selected is, $n = 3$. The last equation is, $S(A)$ gives us the final transformed sequences.

$$A(f) = Amplitude(\mathfrak{F}(A)) \tag{6.1}$$

$$P(f) = Phrase(\mathfrak{F}(A)) \tag{6.2}$$

$$L(f) = log(A(f)) \tag{6.3}$$

$$AL(f) = h_q(f).L(f) \tag{6.4}$$

$$R(f) = L(f) - AL(f) \tag{6.5}$$

$$S(A) = ||\mathfrak{F}^{-1}(exp(R(f) + iP(f)))|| \tag{6.6}$$

Figure 6.5 shows the spectral residual algorithm applied to the original signal values shown in Figure 6.2. In Figure 6.2, the plateau attack only appears on the second signal from the top. In Figure 6.5, the same signal attack window is more salient than the rest of the signals. But one thing we can see from the figure is that the areas around the middle of the attack are not as salient as the corners.

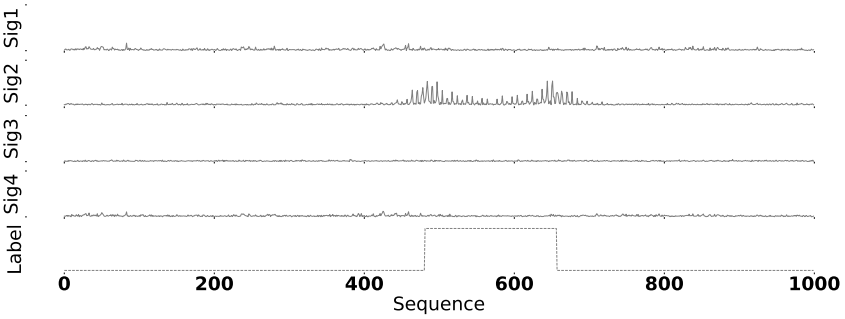


Figure 6.5: Saliency residual of plateau attack for Arbitration ID 10

Especially when the attack window is very long, the saliency seems to disappear giving no information about an attack. To overcome this issue, we changed our label to only the start and finish of the attack. When calculating $R(f)$ in the spectral residual equation, we used n as a moving average size. If the attack starts at signal s, we label $2n + 1$ signals as an attack. That is n signals before the attack starts, 1 right when the attack starts and n for the three consecutive attack signals. The same labeling strategy is used when the attack finishes too. $2n + 1$ labels, n attacks while we are still in the attack window, 1 for the last attacked signal, and n for the upcoming signals. The reason we labeled n benign signals as an attack is due to the moving average size we selected, n, that will affect n benign signals' saliency value.

6.4 Experimental results and discussion

In this section, we show the experimental results of both datasets and a discussion of the results.

6.4.1 Datasets

The RAW dataset we used in the experiment has a total of 17.5 million frames. Using HAMD, we have identified the arbitration IDs and the counter locations in the CAN frames. The dataset contains 27 arbitration IDs, of which 9 IDs have 2 or more counter bits. In this dataset, we have managed to detect attacks by only following the counter bits of the arbitration IDs listed in Table 6.3.

Table 6.3: Arbitration IDs with segmented counter bits

Arbitration ID	Counter bits
0002	[52,53,54,55]
0130	[52,53,54,55]
0131	[52,53,54,55]
0140	[52,53,54,55]
0260	[58,59]
02a0	[1,2]
02b0	[36,37,38,39]
0329	[0,1]
0350	[[16,17,18],[56,57,58,59]]

The PHY dataset has 4 training data that aggregates to 29,669,723 CAN frames and testing data for the plateau attack with 2,150,053 frames. As mentioned earlier, the training dataset has no attack frames. We prepared the attack frames on the training data by studying how attacks are prepared in the test data. The final training dataset is shown in Table 6.4. In the table, the arbitration ID column is the frame arbitration ID, attack# is the number of plateau attacks, and benign# is the number of benign frames. Using these datasets, we trained a U-CAN model that correctly segments attack signals from non-attack signals. In prediction, a window of 128 CAN signals is classified as an attack even if a single signal is segmented as an attack in the window. For each of the arbitration IDs, we used 80% of the data shown in Table 6.4 for training, and the rest for validation to tune the hyperparameters. No testing data is used from the simulation; we used the plateau attack test data to test the trained U-CAN.

6.4.2 Performance evaluation metrics

To evaluate the performance of U-CAN, we used confusion metrics. The confusion matrix we created has true a positive rate (TPR), a false positive rate (FPR), a false negative rate (FNR), and a true negative rate (TNR), each of which is calculated using equations 6.7, 6.8, 6.9, and 6.10 respectively. We have also added the F_1 *Score*, equation 6.11, to compare the performance of the proposed method

Table 6.4: PHY dataset's training data

Arbitration ID	Attack#	Benign#	Total
ID 1	988,745	3,151,074	4,139,819
ID 2	492,530	1,577,613	2,070,143
ID 3	989,255	3,150,571	4,139,826
ID 4	330,489	1,049,608	1,380,097
ID 5	985,057	3,154,763	4,139,820
ID 6	492,050	1,578,094	2,070,144
ID 7	984,140	3,155,675	4,139,815
ID 8	987,421	3,152,397	4,139,818
ID 9	495,600	1,574,544	2,070,144
ID 10	329,708	1,050,389	1,380,097
	Total		29,669,723

with Rec-CNN [92] and ResNet-based [17] methods.

$$TPR = \frac{TP}{TP + FN} \tag{6.7}$$

$$FPR = \frac{FP}{FP + TN} \tag{6.8}$$

$$FNR = \frac{FN}{FN + TP} \tag{6.9}$$

$$TNR = \frac{TN}{TN + FP} \tag{6.10}$$

$$F_1\ Score = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{6.11}$$

In equations 6.7, 6.8, 6.9, and 6.10, TP (True Positive) is for correctly detected attacks, FN (False Negative) is for misclassified attacks, FP (False Positive) is for benign classified as attacks, and TN (True Negative) is for correctly classified benign. On top of the confusion matrix, we have also used ROC (receiver operating characteristic curve) to show the prediction capabilities of U-CAN.

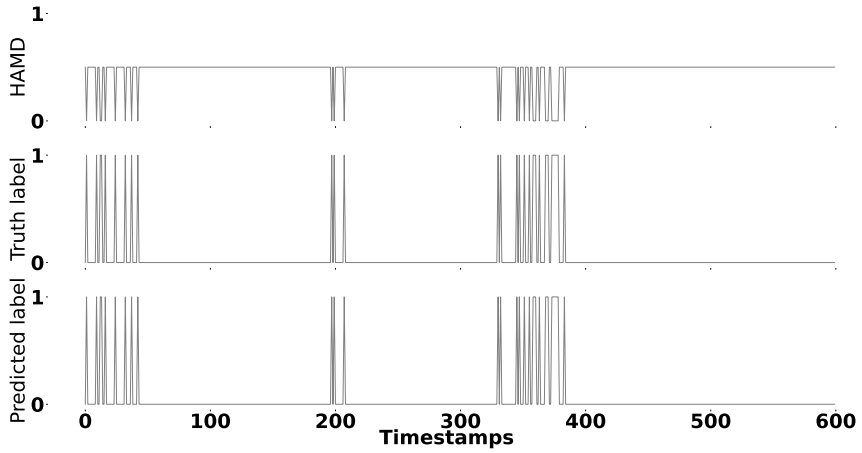


Figure 6.6: HAMD segmentation using U-CAN for RAW datasets

6.4.3 U-CAN training

We trained U-CAN using Tensorflow [93], a machine learning library. We first selected network training hyperparameters using Hyperband. From the list of hyperparameters in Table 6.1, hyperband found the best performing model with values; batch size: 32, learning rate: 0.0001, learning decay: 0.9, learning steps: 10000, scheduler: Exponential Decay, and optimizer: Ftrl. Using the selected hyperparameters, we trained U-CAN for 343 epochs with an early stopping callback that monitors the validation loss with the patience of 30 epochs.

For both datasets, the input has a period of 3 and a sequence length of 128. This means that 3 sets of 128 sequences of values are used during single-step training. In the testing, each of these sequences is segmented into either attack or non-attack. A segment is classified as an attack window if even one signal is segmented as an attack. Figures 6.6 and 6.7, show the RAW and PHY dataset segmentation results, respectively. In the RAW dataset, U-CAN does nothing special except select HAMD values that deviate from the HAMD constant value. The U-CAN in the PHY dataset has one more step of saliency detection before inserting the values for prediction. It only detects the start and finish of attacks, unlike the U-CAN used for the RAW dataset.

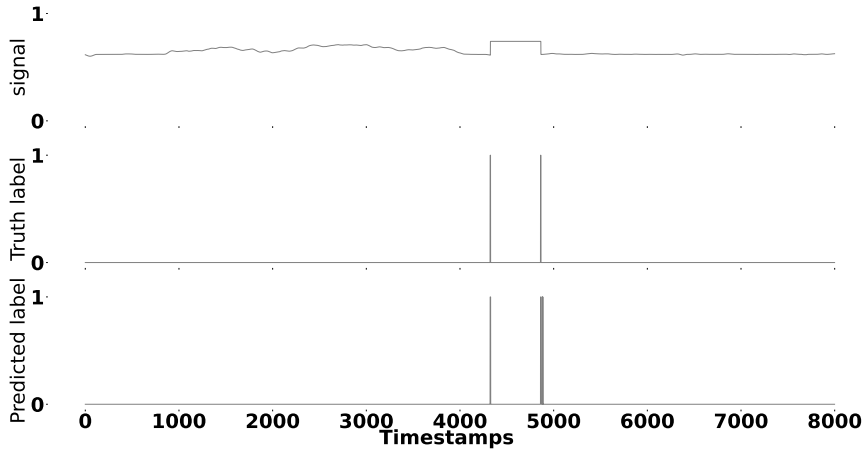


Figure 6.7: Arbitration ID 1's first signal segmentation using U-CAN prediction for PHY datasets

6.4.4 Evaluation

We checked the RAW dataset first with a rule-based system. The rule-based system initially collects CAN frames for 5 seconds. This collection of frames is then separated with respect to the frame arbitration IDs. For each arbitration ID, a HAMD is performed to each bit of the frames. Then, the HAMD value of the last counter bit is checked in the window. If, for instance, we take arbitration ID 0002, the HAMD value of bit 55 should be 0.5 in a window of 5 seconds, as this bit value flips every time it appears in the CAN bus. Similarly, the rest of the arbitration ID's last bit is compared with the HAMD value of 0.5. This time window is labeled as an attack if any of the HAMD values are different from 0.5. In doing so, we have found the results shown in Table 6.5 for all the attack types.

We have compared these results with the currently available CNN-based IDS: a Rec-CNN method [92] and the paper that released the original dataset, and a ResNet-based method [17] using an F_1 *Score*. The rule-based system's F_1 *Score* is 0.997, which is slightly lower than the ReC-CNN method with an F_1 *Score* of 0.999 and slightly higher than the ResNet-based method with an F_1 *Score* of 0.991. This result helps us to only show the comparable results of the methodologies, but in reality, comparison can be troublesome since the rule-based system uses no training data unlike the other two. U-CAN for RAW dataset only segments

Table 6.5: RAW dataset testing results

Data type		Benign	Attack
Normal	Benign	1.0	0.0
	Attack	0.0	0.0
Dos	Benign	0.972	0.038
	Attack	0.0	1.0
Fuzzy	Benign	0.950	0.05
	Attack	0.0	1.0
RPM	Benign	0.950	0.05
	Attack	0.0	1.0
gear	Benign	0.970	0.03
	Attack	0.0	1.0

Table 6.6: RAW dataset U-CAN segmentation results

	Benign	Attack
Benign	1.0	0
Attack	0	1.0

areas where the HAMD has deviated from its actual value. It does this with 100% accuracy in Table 6.6. This implies the result is exactly the same as the rule-based for the whole training data.

For the PHY dataset, Figure 6.8 shows the ROC of the plateau attack detection for all the arbitration IDs. From the figure, we can see that U-CAN accurately detects all the attacks except for two arbitration IDs; arbitration ID 2 and arbitration ID 9. Segmentation in these IDs doesn't work well as in the other arbitration IDs due to the type of signal generated by these arbitration IDs. Some signal sequences resemble a plateau attack. Table 6.7 also shows the confusion matrix results of the detection.

Similarly, we compared the U-CAN to the CANeT [20], the most recent study on the use of CAN physical data, as well as the original publication that provided the PHY dataset. Table 6.8 shows the TPR-to-TNR ratio of U-CAN and CANeT. U-CAN detects plateau attacks better than CANeT does.

Table 6.7: PHY dataset U-CAN attack detection results

Arbitration ID		**Attack**	**Benign**
ID 1	Attack	1.0	0.0
	Benign	0.001	0.999
ID 2	Attack	0.885	0.115
	Benign	0.003	0.997
ID 3	Attack	1.0	0.0
	Benign	0.001	0.999
ID 4	Attack	1.0	0.0
	Benign	0.0	1.0
ID 5	Attack	0.914	0.096
	Benign	0.0	1.0
ID 6	Attack	0.909	0.091
	Benign	0.001	0.999
ID 7	Attack	1.0	0.0
	Benign	0.0	1.0
ID 8	Attack	1.0	0.0
	Benign	0.001	0.999
ID 9	Attack	1.0	0.0
	Benign	0.005	0.995
ID 10	Attack	1.0	0.0
	Benign	0.004	0.996

Table 6.8: TPR / TNR comparison of U-CAN and CANET for plateau attack detection

Method	**True Positive Rate / True Negative Rate**
U-CAN	0.971 / 0.998
CANeT	0.885 / 0.993

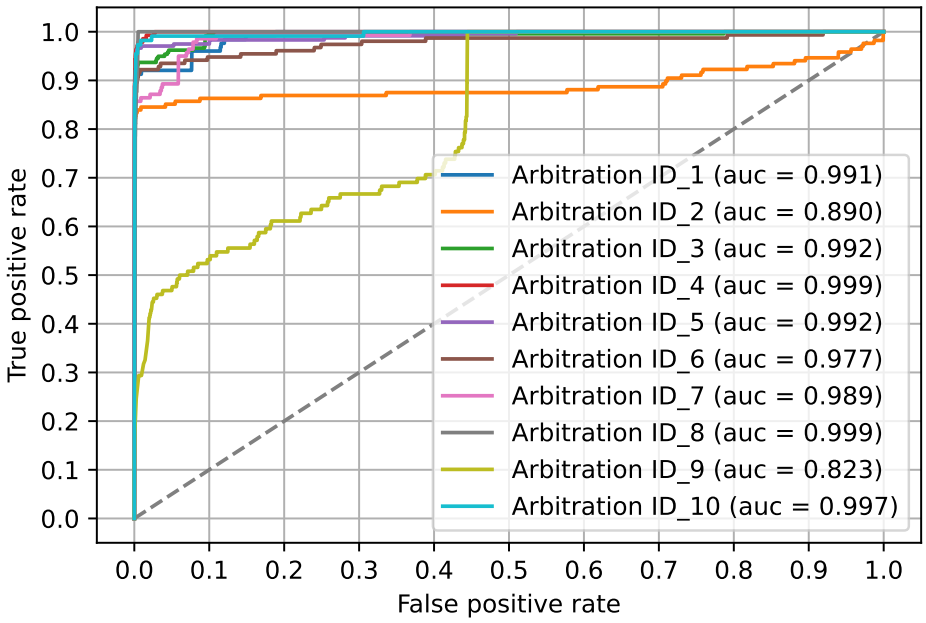


Figure 6.8: ROC of plateau attack detection for all arbitration IDs in PHY dataset

6.5 Conclusion

The number of ECUs that interact in CAN bus has surpassed 100 in modern vehicles. Each of these ECUs adds special features to the vehicles while exposing the vehicles to cyber threats. As a countermeasure for the security exposures, we propose an in-vehicle network attack detection method using convolutional neural networks called U-CAN. U-CAN can be applied to both raw and reverse-engineered CAN frames. Training U-CAN for the raw data first involves calculating the HAMDs of CAN frames. Making use of the HAMD of CAN frame counter-bits, we have created a rule-based IDS system that follows the counter-bits. Any deviation from the constant HAMD value of a counter-bit is flagged as an attack in the rule-based system. We have also used the distribution to train U-CAN that segments these deviations with 100% accuracy. Moreover, we trained U-CAN on a reverse-engineered CAN packets that outperform a similar methodology in this field with a TPR-to-TNR ratio of 0.971/0.998.

Most cars that use CAN have the counter-bits in their CAN frames, but not all

do. There can also be cases where the counter-bits will change at unknown times. Our system will flag this unknown bit flip in the counter bits as an attack. As a continuation for this work, one might improve the rule-based system by studying the cause of these bit flips as well as testing U-CAN on more types of attacks instead of only plateau attacks.

7 Conclusion and future work

This is the thesis' last chapter, which summarizes the entire work in parts. Later, we discuss the research's future path in terms of using deep learning for in-vehicle networks.

7.1 Summary of contributions

The research's main goal is developing a robust IDS that can detect attacks in in-vehicle networks. Deep neural network architectures are trained using the CAN bus data to detect attacks that are inflicted on the CAN bus. Even though such systems can detect the majority of the attacks, they still have certain limitations.

As a first work, LSTM-IDS is proposed to detect in-vehicle network attacks. Dumping CAN frames will show timestamp information, Arbitration ID, and CAN data with other irrelevant information for this research. The timestamp information is temporal information of the CAN frame. Researchers have used this statistically studied this information to create an in-vehicle network IDS. The issue with this type of IDS is that if statistical information does not deviate from its normal range, attacks will not be detected. This is the time where LSTM-IDS comes as a solution for detecting attacks beyond the statics-based methods. LSTM-IDS uses the arbitration IDs of CAN frames to train a deep neural network. The trained LSTM model is expected to learn the sequence of arbitration IDs in the CAN bus. If the LSTM network has managed to learn the sequence of arbitration IDs, assuming that there will be, it can be used to predict a sequence of arbitration IDs in the CAN bus. Attacks in this method will be flagged if the predicted arbitration ID and the true arbitration ID (Arbitration ID that has actually appeared in the CAN bus) are not the same. Since the last layer of this network architecture is Softmax activated, it will give the highest probability to

the most probable class. If the most probable class is the wrong class, LSTM-IDS will flag an attack in the CAN bus. Relying only on the Softmax output of the network architecture might not be the smartest way for multi-class classification. The actual ID might appear in the top few high probable classes. The devised approach of anomaly signal takes this into consideration by using an aggregated log loss. It indicates how good or bad the prediction results are by considering all available classes. This improves a previous work by [15] that also uses arbitration IDs to train a single transition matrix. The transition matrix will hold information about the possible transitions between two different IDs. LSTM-IDS performs better than the transition matrix-based method as it is impossible to grab millions of arbitration ID sequences in a single transition matrix.

Rec-CNN, like the LSTM-IDS, trains a neural network using the arbitration IDs of CAN frames. One of the reasons we utilized LSTM in the earlier work is because it is effective at learning and remembering extended sequences of inputs. However, relying just on the sequence of arbitration ID predictions for intrusion detection will result in lower attack detection. Rec-CNN is a CNN-based IDS for in-vehicle networks that have improved performance over LSTM-IDS. Rec-CNN changes the attack detection problem in the CAN bus to image classification problem. It takes the sequence of arbitration IDs that are preprocessed to an image through recurrence plots. The preprocessing first encodes the arbitration IDs to a numerical value and later is used in the recurrence plot algorithm to create a square image. The images are then used to train a Rec-CNN that has six layers in an arrangement of convolutional, pooling, convolutional, pooling, Dropout, and Dense layers. Each of these layers has hyperparameters that need to be set. These hyperparameters are set to an optimal value through a hyperparameter search algorithm called Hyperband [79]. The final trained Rec-CNN takes an input of 128 sequence of arbitration IDs, that are converted to an image in the preprocessing stage, and classifies it to an attack or non-attack image. In this approach, Rec-CNN has improved performance in comparison to the state-of-the-art research called Inception-ResNet-based [17].

The main drawback with the previous two methods is that in-vehicle network attacks are detected if only the attacks affect the sequence of arbitration ID. If the sequence of the arbitration IDs is not affected, the previous two methods will

fail to detect such attack. The solution for this issue would be to create an LSTM model that predicts the CAN data instead of the arbitration IDs. CAN data is multi-dimensional data and our model needs to handle this property. MLIDS is capable of handling the high-dimensional CAN data without the need for reverse engineering. MLIDS builds one LSTM network architecture for each corresponding arbitration ID. The LSTM network architecture contains an embedding layer, LSTM layer, concatenating layer, and dense layer. For an 8-byte CAN data, the embedding layer takes an input with the shape of $[64, 1]$. The output from this layer is then fed to the LSTM layer. The results from all the bits of the CAN data are then concatenated and used as an input to the last layer. The last layer then provides the predicted CAN frame for this arbitration ID. This is done for all CAN frames grouped by the arbitration IDs. An anomaly signal is calculated during testing by aggregating the loss of all the predictions. With a predefined threshold, if the loss of the window is above it, it will be flagged as an attack. In this way, MLIDS is tested on insertion, drop, and fuzzy attacks.

Training LSTM for high-dimensional CAN bus data can be problematic due to the high number of hyperparameters that need to be tuned. U-CAN comes in handy in managing the high-dimensional CAN bus data. U-CAN is an extension of the image segmentation model with encoder, decoder, and segment classifier modules. The encodes module takes a single arbitration ID's raw CAN data or reverse engineered CAN signals and encodes it through 12 blocks of convolutional layer, batch normalization layer, and a max-pooling layer. The decoder then gets the output from the encoder and computes it through 12 blocks of up-sampling layer, convolutional layer, batch normalization layer, and concatenation layer. On top of the outputs from the encoders, residual connections collected from the encoder are concatenated to each block in the decoder. The resulting values finally pass through a segment classifier module that classifies each signal of the CAN bus to attack or non-attack signal. In such a way, U-CAN has been tested in two datasets.

7.2 Future work

This dissertation primarily focuses on CAN bus intrusion detection systems. The majority of the approaches discussed concentrate on CAN data analysis to learn an attack-free CAN communication pattern. As a result, any deviation from this established pattern is signaled as an attack. However, there are still some open issues that this research can be extended in the future. We outline potential future study trajectories in this section that may be developed upon the foundation of this dissertation.

7.2.1 Inherent machine learning issues

Applying machine learning to solve real-world problems can fail catastrophically in deployment when the distribution of data suddenly shifts. This is usually caused when machine learning practitioners fail to investigate the data used for training. Two types of distribution shifts can occur while training machine learning algorithms for an IDS in in-vehicle networks: covariance shift, and label shift [94]. Covariance shift occurs when the distribution of inputs changes over time without a change in the label function. This for instance can happen in testing a cat and dog classifier that is trained on actual photos of cats and dogs but tested on cartoons of cats and dogs. Similar to this, in in-vehicle networks, an IDS model might have been trained on an old ECU, but through time the ECU might be updated through the Update over the air (OTA) [95]. The label shift opposite to the covariance shift occurs when there is a shift in the label while the input distribution is the same. In the LSTM-IDS, a sequence of arbitration IDs is learned through the LSTM. But these same UOA can cause a change in the sequence of arbitration IDs resulting in a label shift. These updates can cause how frames used to appear in the CAN bus to create a distribution shift. Therefore, one needs to retrain IDS models each time an update is made in ECUs. In the future, we will work on finding ways to update the models in a distributed fashion capable of connecting to the in-vehicle network IDS systems.

Performance of deep learning is significantly influenced by the training data. Deep learning won't be able to generalize as well if there aren't enough instances, which will result in incorrect inferences. The dataset we used in the experiment

was acquired under normal conditions, with no noises or event-triggered situations taken into account. Consider the ABS control system of the car, which only kicks in during emergencies. Due to the absence of information about these occurrences in the dataset used, it is difficult for the trained model to classify it as a non-attack. To train the models for these circumstances, we lack the essential data. Since anything that deviates from the training data would cause deep learning models to fail, the IDSs might classify these situations as an attack. Hence, in the future a deep analysis of these scenarios would be considered.

7.2.2 Low computational capability of the CAN bus

Apart from the inherent machine learning problems, there can also be computational resource constraints in designing sophisticated machine learning algorithms in in-vehicle network IDS systems. CAN bus has a low data bitrate and one needs to implement the IDS system outside of the network. Our work listens in to the CAN bus and does the inferences in Jetson TX2 [85]. This can also be a limitation to the research as it will incur additional costs. In the future, we will study how the machine learning models can be resource-efficient.

7.2.3 Extension of this work to other CAN-based systems

The CAN bus security on vehicles is the major focus of the dissertation. To train the methods, our work just pulls training data from a vehicle. However, CAN is utilized in a variety of systems, including those in robotics [96], aircraft [97], and industrial automation [98]. Any system that makes use of the CAN bus can expand upon our work. However, utilizing it the same way as it is in vehicles could not work. One must comprehend the precise usage of CAN in other systems in order to apply the approaches given in this work to other CAN applications. It's possible that the network structure of the CAN in in-vehicles differs from that of other systems. If we take LSTM-IDS and Rec CNN as an example, they both examine the patterns of arbitration IDS in CAN frames. We are anticipated to comprehend how arbitration IDS of CAN frames are handled in the other CAN systems if these two works are to be utilised.

Millions of training data must be gathered in order to properly train deep

learning neural networks. If deep learning is to be applied in these systems, it is important to thoroughly research the CAN frame collection process. As already indicated, the vehicles' OBD II ports are used to gather the data. But would the other systems have similar entry points. The learning of the CAN data patterns is another aspect of our work. However, it's possible that the CAN data component of the CAN frames will differ from the in-vehicle networks, necessitating greater preprocessing. In order to expand this work to additional CAN systems, we must completely comprehend how ECUs are connected to the CAN bus and how the arbitration IDs of CAN frames are handled in the networks.

7.2.4 Post attack detection stage

The attack detection step is the sole topic covered in the dissertation since we felt it to be the most important one. The post-detection phases are still up for debate. How should the defense system respond to intrusions? Should we alert the driver about attacks and allow them take any necessary action? or let the system proactively take action to avert the disasters that might be caused by attacks?

The status of the in-vehicle networks may be shown to the driver using just a display or an alarm system, making notification of the driver relatively simpler. However, this option would be unavailable given the impending arrival of fully autonomous vehicles. Few post-detection techniques are discussed in [58]. One approach is to use secure patches to reset a corrupted ECU to its uninfected state, but this requires us to identify the affected ECU first. This would be effective if the approaches presented here could recognize the hacked ECUs. This problem will still need to be addressed in further work.

Acknowledgements

Thank you, and thank you.

References

[1] Robert N. Charette. How software is eating the car, 2011. https://spectrum.ieee.org/software-eating-car, Accessed: 2022-02-01.

[2] Robert Bosch GmbH. Bosch can specification version 2.0, 1991.

[3] Karl Henrik Johansson, Martin Törngren, and Lars Nielsen. Vehicle applications of controller area network. In *Handbook of networked and embedded control systems*, pages 741–765. Springer, 2005.

[4] Mamdooh Al-Saud, Ali M Eltamaly, Mohamed A Mohamed, and Abdollah Kavousi-Fard. An intelligent data-driven model to secure intravehicle communications based on machine learning. *IEEE Transactions on Industrial Electronics*, 67(6):5112–5119, 2019.

[5] Shengtuo Hu, Qi Alfred Chen, Jiwon Joung, Can Carlak, Yiheng Feng, Z Morley Mao, and Henry X Liu. Cvshield: Guarding sensor data in connected vehicle with trusted execution environment. In *Proceedings of the Second ACM Workshop on Automotive and Aerial Vehicle Security*, pages 1–4, 2020.

[6] Omid Avatefipour and Hafiz Malik. State-of-the-art survey on in-vehicle network communication (can-bus) security and vulnerabilities. *arXiv preprint arXiv:1802.01725*, 2018.

[7] Dakshnamoorthy Manivannan, Shafika Showkat Moni, and Sherali Zeadally. Secure authentication and privacy-preserving techniques in vehicular ad-hoc networks (vanets). *Vehicular Communications*, 25:100247, 2020.

[8] Robert Buttigieg, Mario Farrugia, and Clyde Meli. Security issues in controller area networks in automobiles. In *2017 18th International Conference on*

Sciences and Techniques of Automatic Control and Computer Engineering (STA), pages 93–98. IEEE, 2017.

[9] Sen Nie, Ling Liu, Yuefeng Du, and Wenkai Zhang. Over-the-air: How we remotely compromised the gateway, bcm, and autopilot ecus of tesla cars. *Briefing, Black Hat USA*, 2018.

[10] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015(S 91), 2015.

[11] Juan Deng, Lu Yu, Yu Fu, Oluwakemi Hambolu, and Richard R Brooks. Security and data privacy of modern automobiles. In *Data Analytics for Intelligent Transportation Systems*, pages 131–163. Elsevier, 2017.

[12] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462. IEEE, 2010.

[13] Jihas Khan. Vehicle network security testing. In *2017 Third International Conference on Sensing, Signal Processing and Security (ICSSS)*, pages 119–123. IEEE, 2017.

[14] Kazuki Iehira, Hiroyuki Inoue, and Kenji Ishida. Spoofing attack using bus-off attacks against a specific ecu of the can bus. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–4. IEEE, 2018.

[15] Mirco Marchetti and Dario Stabili. Anomaly detection of can bus messages through analysis of id sequences. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1577–1583. IEEE, 2017.

[16] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:pages 354–377, 2018.

[17] Hyun Min Song, Jiyoung Woo, and Huy Kang Kim. In-vehicle network intrusion detection using deep convolutional neural network. *Vehicular Communications*, 21:100198, 2020.

[18] Jean-Pierre Eckmann, S Oliffson Kamphorst, David Ruelle, et al. Recurrence plots of dynamical systems. *World Scientific Series on Nonlinear Science Series A*, 16:441–446, 1995.

[19] Eunbi Seo, Hyun Min Song, and Huy Kang Kim. GIDS: Gan based intrusion detection system for in-vehicle network. In *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, pages 1–6. IEEE, 2018.

[20] Markus Hanselmann, Thilo Strauss, Katharina Dormann, and Holger Ulmer. CANet: An unsupervised intrusion detection system for high dimensional CAN bus data. *IEEE Access*, 8:58194–58205, 2020.

[21] Craig Smith. *The car hacker's handbook: a guide for the penetration tester*. No Starch Press, 2016.

[22] Trupil Limbasiya, Amrita Ghosal, and Mauro Conti. Autosec: Secure automotive data transmission scheme for in-vehicle networks. In *23rd International Conference on Distributed Computing and Networking*, pages 208–216, 2022.

[23] Charlie Miller and Chris Valasek. Adventures in automotive networks and control units. *Def Con*, 21(260-264):15–31, 2013.

[24] Nasser Nowdehi, Aljoscha Lautenbach, and Tomas Olovsson. In-vehicle CAN message authentication: An evaluation based on industrial criteria. In *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*, pages 1–7. IEEE, 2017.

[25] Emad Aliwa, Omer Rana, Charith Perera, and Peter Burnap. Cyberattacks and countermeasures for in-vehicle networks. *ACM Computing Surveys (CSUR)*, 54(1):1–37, 2021.

[26] CAN Specification. Bosch. *Robert Bosch GmbH*, 1991.

[27] Hyunsung Lee, Seong Hoon Jeong, and Huy Kang 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)*, pages 57–5709. IEEE, 2017.

[28] Sibylle Fröschle and Alexander Stühring. Analyzing the capabilities of the can attacker. In *European Symposium on Research in Computer Security*, pages 464–482. Springer, 2017.

[29] Wufei Wu, Renfa Li, Guoqi Xie, Jiyao An, Yang Bai, Jia Zhou, and Keqin Li. A survey of intrusion detection for in-vehicle networks. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):919–933, 2019.

[30] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX Security Symposium (USENIX Security 11)*, 2011.

[31] Mert D Pesé, Jay W Schauer, Junhui Li, and Kang G Shin. S2-can: Sufficiently secure controller area network. In *Annual Computer Security Applications Conference*, pages 425–438, 2021.

[32] Patrizio Pelliccione, Eric Knauss, S Magnus Ågren, Rogardt Heldal, Carl Bergenhem, Alexey Vinel, and Oliver Brunnegård. Beyond connected cars: A systems of systems perspective. *Science of Computer Programming*, 191: 102414, 2020.

[33] Mehmet Bozdal, Mohammad Samie, Sohaib Aslam, and Ian Jennions. Evaluation of can bus security challenges. *Sensors*, 20(8):2364, 2020.

[34] Arash Shaghaghi, Mohamed Ali Kaafar, and Sanjay Jha. Wedgetail: An intrusion prevention system for the data plane of software defined networks. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 849–861, 2017.

[35] Roland Kammerer, Bernhard Frömel, and Armin Wasicek. Enhancing security in CAN systems using a star coupling router. In *7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*, pages 237–246. IEEE, 2012.

[36] Sebastian Bittl. Attack potential and efficient security enhancement of automotive bus networks using short macs with rapid key change. In *International Workshop on Communication Technologies for Vehicles*, pages 113–125. Springer, 2014.

[37] Assaf Harel and Amir Hezberg. Optimizing CAN bus security with in-place cryptography. Technical report, 2019.

[38] Adam Hanacek and Martin Sysel. Design and implementation of an integrated system with secure encrypted data transmission. In *Computer Science On-line Conference*, pages 217–224. Springer, 2016.

[39] Qiyan Wang and Sanjay Sawhney. VeCure: A practical security framework to protect the CAN bus of vehicles. In *2014 International Conference on the Internet of Things (IOT)*, pages 13–18. IEEE, 2014.

[40] Bogdan Groza, Stefan Murvay, Anthony Van Herrewege, and Ingrid Verbauwhede. Libra-can: a lightweight broadcast authentication protocol for controller area networks. In *International Conference on Cryptology and Network Security*, pages 185–200. Springer, 2012.

[41] Anthony Van Herrewege, Dave Singelee, and Ingrid Verbauwhede. Canauth-a simple, backward compatible broadcast authentication protocol for can bus. In *ECRYPT workshop on Lightweight Cryptography*, volume 2011, page 20. ECRYPT, 2011.

[42] Ryo Kurachi, Yutaka Matsubara, Hiroaki Takada, Naoki Adachi, Yukihiro Miyashita, and Satoshi Horihata. Cacan-centralized authentication system in can (controller area network). In *14th Int. Conf. on Embedded Security in Cars (ESCAR 2014)*, 2014.

[43] Andreea-Ina Radu and Flavio D Garcia. Leia: A lightweight authentication protocol for can. In *European Symposium on Research in Computer Security*, pages 283–300. Springer, 2016.

[44] Ki-Dong Kang, Youngmi Baek, Seonghun Lee, and Sang Hyuk Son. An attack-resilient source authentication protocol in controller area network. In *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 109–118. IEEE, 2017.

[45] Samir Fassak, Younes El Hajjaji El Idrissi, Noureddine Zahid, and Mohamed Jedra. A secure protocol for session keys establishment between ecus in the can bus. In *2017 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, pages 1–6. IEEE, 2017.

[46] Bogdan Groza and Stefan Murvay. Efficient protocols for secure broadcast in controller area networks. *IEEE Transactions on Industrial Informatics*, 9(4): 2034–2042, 2013.

[47] Giampaolo Bella, Pietro Biondi, Gianpiero Costantino, and Ilaria Matteucci. Toucan: A protocol to secure controller area network. In *Proceedings of the ACM Workshop on Automotive Cybersecurity*, pages 3–8, 2019.

[48] Dennis K Nilsson, Ulf E Larson, and Erland Jonsson. Efficient in-vehicle delayed data authentication based on compound message authentication codes. In *2008 IEEE 68th Vehicular Technology Conference*, pages 1–5. IEEE, 2008.

[49] Zhaojun Lu, Qian Wang, Xi Chen, Gang Qu, Yongqiang Lyu, and Zhenglin Liu. LEAP: A lightweight encryption and authentication protocol for in-vehicle communications. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 1158–1164. IEEE, 2019.

[50] Ali Shuja Siddiqui, Yutian Gui, Jim Plusquellic, and Fareena Saqib. Secure communication over CANBus. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1264–1267. IEEE, 2017.

[51] Tri P Doan and Subramaniam Ganesan. CAN crypto FPGA chip to secure data transmitted through CAN FD bus using AES-128 and SHA-1 algorithms with a symmetric key. Technical report, SAE Technical Paper, 2017.

[52] Mabrouka Gmiden, Mohamed Hedi Gmiden, and Hafedh Trabelsi. An intrusion detection method for securing in-vehicle can bus. In *2016 17th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, pages 176–180. IEEE, 2016.

[53] Yujing Wu, Yeon-Jin Kim, Zheyan Piao, Jin-Gyun Chung, and Yong-En Kim. Security protocol for controller area network using ecandc compression algorithm. In *2016 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, pages 1–4. IEEE, 2016.

[54] Wael A Farag. CANTrack: Enhancing automotive CAN bus security using intuitive encryption algorithms. In *2017 7th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO)*, pages 1–5. IEEE, 2017.

[55] Luca Dariz, Michele Selvatici, Massimiliano Ruggeri, Gianpiero Costantino, and Fabio Martinelli. Trade-off analysis of safety and security in can bus communication. In *2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pages 226–231. IEEE, 2017.

[56] Clarence Chio and David Freeman. *Machine learning and security: Protecting systems with data and algorithms.* " O'Reilly Media, Inc.", 2018.

[57] Kyong-Tak Cho and Kang G Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 911–927, 2016.

[58] Wonsuk Choi, Kyungho Joo, Hyo Jin Jo, Moon Chan Park, and Dong Hoon Lee. VoltageIDS: Low-level communication characteristics for automotive intrusion detection system. *IEEE Transactions on Information Forensics and Security*, 13(8):2114–2129, 2018.

[59] Swarup Mohalik, AC Rajeev, Manoj G Dixit, S Ramesh, P Vijay Suman, Paritosh K Pandya, and Shengbing Jiang. Model checking based analysis of end-to-end latency in embedded, real-time systems with clock drifts. In *Proceedings of the 45th annual Design Automation Conference*, pages 296–299, 2008.

[60] Ki-Dong Kang, Youngmi Baek, Seonghun Lee, and Sang H Son. An analysis of voltage drop as a security feature in controller area network. In *Proceedings of the 2016 IEMEK Symposium on Embedded Technology, 216AD, Daejeon, Korea*, pages 26–27, 2016.

[61] Tsvika Dagan and Avishai Wool. Parrot, a software-only anti-spoofing defense system for the can bus. *ESCAR EUROPE*, 34, 2016.

[62] Mee Lan Han, Jin Lee, Ah Reum Kang, Sungwook Kang, Jung Kyu Park, and Huy Kang Kim. A statistical-based anomaly detection method for connected cars in internet of things environment. In *International Conference on Internet of Vehicles*, pages 89–97. Springer, 2015.

[63] Adrian Taylor, Nathalie Japkowicz, and Sylvain Leblanc. Frequency-based anomaly detection for the automotive can bus. In *2015 World Congress on Industrial Control Systems Security (WCICSS)*, pages 45–49. IEEE, 2015.

[64] Yoshihiro Hamada, Masayuki Inoue, Hiroshi Ueda, Yukihiro Miyashita, and Yoichi Hata. Anomaly-based intrusion detection using the density estimation of reception cycle periods for in-vehicle networks. *SAE International Journal of Transportation Cybersecurity and Privacy*, 1(11-01-01-0003):39–56, 2018.

[65] Congli Ling and Dongqin Feng. An algorithm for detection of malicious messages on can buses. In *2012 national conference on information technology and computer science. Atlantis Press*, volume 10. Citeseer, 2012.

[66] Dario Stabili, Mirco Marchetti, and Michele Colajanni. Detecting attacks to internal vehicle networks through hamming distance. In *2017 AEIT International Annual Conference*, pages 1–6. IEEE, 2017.

[67] Michael Müter and Naim Asaj. Entropy-based anomaly detection for in-vehicle networks. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 1110–1115. IEEE, 2011.

[68] Ching Hsien Hsu, Feng Xia, Xingang Liu, and Shangguang Wang. Internet of vehicles-safe and intelligent mobility. In *Proceedings of the 2nd International Conference, IOV*, volume 2015, pages 19–21. Springer, 2015.

[69] Mirco Marchetti, Dario Stabili, Alessandro Guido, and Michele Colajanni. Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms. In *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pages 1–6. IEEE, 2016.

[70] Shuji Ohira, Araya Kibrom Desta, Ismail Arai, and Kazutoshi Fujikawa. PLI-TDC: Super fine delay-time based physical-layer identification with time-to-digital converter for in-vehicle networks. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 176–186, 2021.

[71] Andreas Theissler. Anomaly detection in recordings from in-vehicle networks. *Big data and applications*, 23:26, 2014.

[72] Min-Ju Kang and Je-Won Kang. A novel intrusion detection method using deep neural network for in-vehicle network security. In *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, pages 1–5. IEEE, 2016.

[73] Adrian Taylor, Sylvain Leblanc, and Nathalie 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)*, pages 130–139. IEEE, 2016.

[74] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. Malware traffic classification using convolutional neural network for representation learning. In *2017 International Conference on Information Networking (ICOIN)*, pages 712–717. IEEE, 2017.

[75] Erxue Min, Jun Long, Qiang Liu, Jianjing Cui, and Wei Chen. TR-IDS: Anomaly-based intrusion detection through text-convolutional neural network and random forest. *Security and Communication Networks*, 2018, 2018.

[76] Ming Zhang, Boyi Xu, Shuai Bai, Shuaibing Lu, and Zhechao Lin. A deep learning method to detect web attacks using a specially designed CNN. In *International Conference on Neural Information Processing*, pages 828–836. Springer, 2017.

[77] Daniel Gibert, Carles Mateu, Jordi Planes, and Ramon Vicens. Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques*, 15(1):15–28, 2019.

[78] Deep Learning. Ian goodfellow and yoshua bengio and aaron courville, 2016.

[79] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[80] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9): 2352–2449, 2017.

[81] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[82] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

[83] Huy Kang Kim. CAR HACKING DATASET. https://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset, 2018. [Online; accessed 25-Sep-2021].

[84] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[85] Nvidia Developer. Nvidia jetson tx2 module, nivida embedded computing, 2018.

[86] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[87] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[88] Daniel Frassinelli, Sohyeon Park, and Stefan Nürnberger. I know where you parked last summer: Automated reverse engineering and privacy analysis of modern cars. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1401–1415. IEEE, 2020.

[89] Mathias Perslev, Sune Darkner, Lykke Kempfner, Miki Nikolic, Poul Jørgen Jennum, and Christian Igel. U-sleep: resilient high-frequency sleep staging. *NPJ digital medicine*, 4(1):1–12, 2021.

[90] Xiaodi Hou and Liqing Zhang. Saliency detection: A spectral residual approach. In *2007 IEEE Conference on computer vision and pattern recognition*, pages 1–8. Ieee, 2007.

[91] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3009–3017, 2019.

[92] Araya Kibrom Desta, Shuji Ohira, Ismail Arai, and Kazutoshi Fujikawa. Rec-cnn: In-vehicle networks intrusion detection using convolutional neural networks trained on recurrence plots. *Vehicular Communications*, page 100470, 2022.

[93] Martín Abadi, Paul Barham, and Jianmin Chen. TensorFlow: A system for Large-Scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA, November 2016. USENIX Association. ISBN 978-1-931971-33-1. URL `https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi`.

[94] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

[95] Kathiresh Mayilsamy, Neelaveni Ramachandran, and Vismitha Sunder Raj. An integrated approach for data security in vehicle diagnostics over internet protocol and software update over the air. *Computers & Electrical Engineering*, 71:578–593, 2018.

[96] Martino Migliavacca, Andrea Bonarini, and Matteo Matteucci. Rtcan-a real-time can-bus protocol for robotic applications. In *ICINCO (2)*, pages 353–360, 2013.

[97] Dirk Dubois. *Evaluating the security of ARINC-825 and controller area networks, the impact of bus security in aerospace*. McGill University (Canada), 2018.

[98] Chapter 19 - controller area network (can). In Kevin M. Lynch, Nicholas Marchuk, and Matthew L. Elwin, editors, *Embedded Computing and Mechatronics with the PIC32*, pages 249–265. Newnes, Oxford, 2016. ISBN 978-0-12-420165-1.

Publication List

Journals

- ***Desta, A.K.**, Ohira, S., Arai, I. and Fujikawa, K., 2020. Long Short-Term Memory Networks for In-Vehicle Networks Intrusion Detection Using Reverse Engineered Automotive Packets. Journal of Information Processing, 28, pp.611-622.
- ***Desta, A.K.**, Ohira, S., Arai, I. and Fujikawa, K., 2022. Rec-CNN: In-vehicle networks intrusion detection using convolutional neural networks trained on recurrence plots. Vehicular Communications, 35, p.100470.
- Ohira, S., **Desta, A.K.**, Arai, I., Inoue, H. and Fujikawa, K., 2020. Normal and malicious sliding windows similarity analysis method for fast and accurate IDS against DoS attacks on in-vehicle networks. IEEE Access, 8, pp.42422-42435.

International conferences

- ***Desta, A.K.**, Ohira, S., Arai, I. and Fujikawa, K., 2020, March. ID sequence analysis for intrusion detection in the CAN bus using long short term memory networks. In 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops) (pp. 1-6). IEEE.
- ***Desta, A.K.**, Ohira, S., Arai, I. and Fujikawa, K., 2020, November. Mlids: Handling raw high-dimensional can bus data using long short-term memory networks for intrusion detection in in-vehicle networks. In 2020 30th International Telecommunication Networks and Applications Conference (ITNAC) (pp. 1-7). IEEE.
- ***Desta, A.K.**, Ohira, S., Arai, I. and Fujikawa, K., 2022. U-CAN: A Convolutional Neural Network Based Intrusion Detection for Controller Area Networks NETSAP 2022 – The 12th IEEE International Workshop on Network Technologies for Security, Administration & Protection (accepted)

- Ohira, S., **Desta, A.K.**, Arai, I. and Fujikawa, K., 2021, May. PLI-TDC: Super Fine Delay-Time Based Physical-Layer Identification with Time-to-Digital Converter for In-Vehicle Networks. In Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (pp. 176-186).

  Ohira, S.,**Desta, A.K.** , Kitagawa, T., Arai, I. and Kazutoshi, F., 2020, July. Divider: Delay-time based sender identification in automotive networks. In 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC) (pp. 1490-1497). IEEE.

N.B. international conferences marked as * are related to this doctoral thesis.