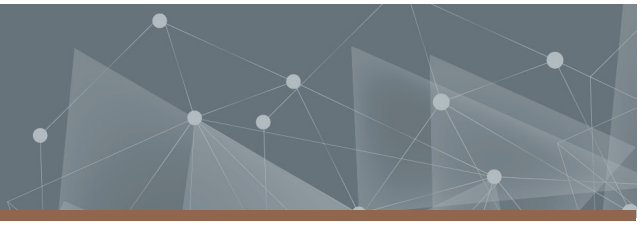
CHALMERS
UNIVERSITY OF TECHNOLOGY



Machine learning for automotive cyber-security

Anomaly detection in CAN

Master's thesis in Complex Adaptive Systems

ELLEN SANDÉN, ARBNOR ZEQIRI

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Machine learning for automotive cybersecurity

Anomaly detection in CAN

ELLEN SANDÉN, ARBNOR ZEQIRI

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2021

Machine learning for automotive cybersecurity
Anomaly detection in CAN
ELLEN SANDÉN, ARBNOR ZEQIRI

Supervisor: Martin Fabian, Department of Electrical Engineering,
Sankar Sathyamoorthy, QRTECH, Christoffer Levandowski, QRTECH
Examiner: Martin Fabian, Department of Electrical Engineering

Master's Thesis 2021
Department ofof Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Machine learning for automotive cybersecurity
Anomaly detection in CAN
ELLEN SANDÉN, ARBNOR ZEQIRI
Department of Electrical Engineering
Chalmers University of Technology

Abstract

As modern cars get increasingly computerized, they become more susceptible to hacker attacks. One of the main attack surfaces in a car is the CAN bus, which is a network that allows all electrical components in the vehicle to communicate with each other. The lack of encryption in CAN makes it vulnerable and in need of an external attack detection system. One way such a system can operate is by learning the normal behavior of the bus, and then analyse the incoming messages to search for anything that diverges from the norm. Any large enough deviation is deemed a potential attack.

In this thesis, a two branched anomaly detection system is developed using several types of machine learning algorithms. The time between the CAN messages, as well as the two main components of the CAN data: the ID and the data load, are analysed in real time on both branches simultaneously to see whether the messages are normal. The branches are combined in two different ways. In the first it is enough for one branch to interpret messages as anomalous to classify them as an attack, whereas in the second both branches have to agree on an attack classification.

To cover as many different types of potential attacks as possible, the two branches are designed to be able to detect different types of anomalies. The first branch consists of a combination of a one-class support-vector machine and a neural network autoencoder, where the former is able to detect anomalies in the time between messages and the latter detects anomalies in the content of the individual data loads. The second branch consists of a neural network autoencoder with convolutional and long short term memory layers, which detects anomalies in the correlations in time between messages, as well as correlations between ID and data load.

By using two different data sets with very different types of simulated attacks, it is argued in this thesis that the combined branches approach makes the finished anomaly detector more robust and secure. It is shown that one branch performs better on the first data set and the other on the second data set, and by combining the branches there is an increased protection toward different types of potential attacks. Separately, the best performing branch on either data set reaches a True Positive Rate of 1.0 and 0.997 respectively, and False Positive Rates of 0.0 and 0.007. Through the combined approach, the anomaly detector is able to reach a best True positive Rate of 1.0 and 0.998 on the two data sets, and a best False Positive Rate of 0.001 and 0.0, depending on the type of combination of the branches.

Acknowledgements

We would like to thank Christoffer Levandowski and Sankar Sathyamoorthy for giving us the opportunity to work on an interesting project, and everyone else at QRTECH for providing a fun and welcoming atmosphere from day one. We would also further like to thank Sankar Sathyamoorthy for much help and valuable inputs, as well as good advice that has allowed us to move forward when we have been stuck. Our Chalmers supervisor and examiner, Martin Fabian, has also been of great help and offered many fruitful discussions as well as useful input on the report.

Ellen Sandén, Arbnor Zeqiri, Gothenburg, June 2021

Contents

1
Introduction

The future of vehicles is one where they are more and more computerized. The reason for this is the pressure on manufacturers to deliver vehicles that are more connected, safer to drive and run more efficiently. Advanced Driver Assistance Systems (ADAS) technologies such as collision detection and automatic parking systems [1], together with developments in the Vehicle-To-Everything (V2X) [2] domain are leading to the increased need of computerization.

In a modern vehicle, the different components are controlled by individual Electronic Control Units (ECUs) [3]. For these to reliably communicate with each other in real time, a fast, lightweight network system is needed. One of the most common communication networks used for this purpose is the CAN (Communication Area Network) bus [4].

Since the CAN bus was designed to be cheap and lightweight, the available bandwidth is limited. This makes message encryption difficult, as any encryption key would be too weak to provide any sort of useful protection. Furthermore, the safety of a vehicle is dependent on in-vehicle-communication being fast, and encryption might cause unacceptable delays [5]. Because of this, CAN communication is completely unencrypted, and can easily be targeted for intrusion.

In [6], researchers show that by using physical connections like the built in OBD-II port, used for diagnostics and updating the ECUs, they are able to completely control safety critical functions of the vehicle such as control the brakes, kill the engine, control the instrument cluster or temporarily boost the engine RPM. Criticism arose that attackers requiring a physical connection to cause a digital malfunction could just as well do it by other physical means. This led to [7] subsequently showing that using the large number of attack surfaces, caused by the increased digitalization mentioned previously, it was possible to achieve the same control remotely. Other examples of hackers being able to completely control a vehicle are [8], [9]. An intruder could also gain access to sensitive information about the driver, such as their address or identity [10], [11].

The unprotected nature of CAN buses thus poses an increasing risk to the safety and privacy of the driver. It also poses a considerable risk to society as a whole. In [12] it is shown that vehicles have been used by terrorists to cause harm to the public. In other words, having vehicles that are robust against security threats is vital for avoiding potential harm to society.

Since the security problem in the CAN bus cannot be solved by introducing encryption in the traffic of the bus, another solution is needed. One conceivable way to tackle the problem is building a separate system that can detect any incoming threats, and subsequently warn the user. Various previous attempts at building such

a system for the CAN bus have been made [5], [13], [14], [15], [16], [17], [18], [19], [20]. Many intrusion detection systems focus on capturing the normal behaviour of the CAN bus, rather than directly finding intrusions. This is because malicious attacks can take practically an infinite number of different forms. If a system focused on detecting one or a few specific types of known attacks, hackers could just come up with a different one to outsmart the system. As such, most detection systems are actually anomaly detectors, rather than intrusion detectors, as the nature of true intrusions is unknown. Using machine learning algorithms as anomaly detectors for CAN is then a viable approach since they can be used to predict the normal behaviour of the CAN traffic. Any large enough deviation from the norm will then be classified as a possible attack.

Among previous works in CAN bus anomaly detection, there are various different approaches taken. Some use frequency based methods, such as in [13], where it is assumed that normal messages in the CAN bus occur in a cyclic manner, with approximately equal amount of time between messages, but that an attacker will inject messages much faster. If time between messages is shorter than a threshold value, it is classified as an attack. This system can thus only detect attacks in the form of rapidly injected messages, and would fail if the injected messages occurred at the same frequency as the normal data.

There are also specification based models, such as [14], where the authors try to find specific rules regarding what an allowed message can look like. Differences from this standard are classified as intrusions.

Statistical methods have also been used, such as in [5], where the authors use a combination of a Hidden Markov Model and a regression model where the former is used to capture the normal behavior of the CAN bus and the latter to train a threshold for what is considered an anomaly.

Information theoretic methods have been used as well, as in [15], where the authors use an information theoretic view of the normal behavior of the CAN bus, where normal entropy is measured and deviations from this indicate an attack. They also use relative entropy of ECUs to determine which ECU is being attacked.

Many works on CAN bus anomaly detection use various kinds of machine learning approaches. Out of these, several different architectures of neural networks have been used, such as convolutional neural networks [16], recurrent networks with Long Short Term Memory units [17], deep feed forward neural networks [18], and generative adversarial networks [19]. A different type of machine learning algorithm, called hierarchical temporal memory, has also been used to capture CAN bus anomalies [20].

However, to the best of our knowledge, none of these previous approaches have included time dependent correlations of both CAN data loads and CAN ID at the same time, but rather treat each ID as independent. Thus, the current work takes anomaly detection in CAN further by capturing correlations in time between different IDs and different data loads, as well as between IDs and data loads.

The number of different types of attacks that can be detected is increased by using a sophisticated architecture consisting of two different branches. The CAN messages go through both branches simultaneously, and two different configurations are used to decide whether or not a message will be deemed an attack. The first

branch will consist of a One-Class Support-Vector Machine [21] that captures time dependencies between the CAN messages, followed by the data load of the CAN messages going through a Neural Network Autoencoder [22]. If at any of these two steps an anomaly is detected, the whole branch will classify the current sequence an anomaly. The second branch will consist of a combined Convolutional, Long Short Term Memory Autoencoder network, where the data is divided into time windows of a certain length, and the convolutional filters will move in the dimensions of ID, data, and time. In this way correlations are captured between all these.

The thesis is structured as follows. First, Chapter 2 gives a background on the theoretical framework for CAN, the machine learning algorithm OCSVM, neural networks and their many variations such as autoencoders, LSTM and CNN. It is followed by Chapter 3 presenting the CAN data set used in the thesis, the details of the neural network architectures, the minutiae of the two branches and how we specifically combine these branches for the problem at hand. Next is Chapter 4, where the results on how well the branches are able to classify whether an anomaly has occurred using ROC and other metrics are presented. The thesis is finished with Chapter 5 where results are reflected upon and where future directions for CAN anomaly detection are proposed.

2
Theory

2.1 CAN

The Communication Area Network (CAN) [4] is a vehicle bus that enables the different ECUs in a vehicle to communicate with each other. Physically, the communication is realized through a two wire bus. The ISO 11898 standard, that defines the CAN bus, describes the physical and the data link layers of the OSI model [23]. The higher levels of the communication protocols, such as the application layer, are vendor and application specific.

CAN works as an open broadcasting medium where each ECU connected to the network can send and receive any message. Each ECU in the network can then decide what messages are relevant to receive. Messages sent by the ECU are in the form of frames. These frames have the following format

- SoF: A '0' in the Start of Frame indicates that a node intends to talk.
- ID: Frame identifier - the lower the ID, the higher the priority of the message.
- RTR: The Remote Transmission Request field is used to specify whether a node sends or requests data from another node.
- Control: Contains the Identifier Extension Bit (IDE) bit. This bit is '0' if an 11-bit ID and '1' if 29-bit ID is used. It also contains Data Length Code (DLC) that specifies the length of the data load (0 to 8 bytes).
- Data: Contains the data load of the message.
- CRC: The Cyclic Redundancy Check ensures the integrity of the data.
- ACK: Indicates whether the node has correctly received and acknowledged the data.
- EOF: Marks the end of the CAN frame.

From the list above, it is seen that the ID serves both as an identifier of a message and as an arbitration mechanism.

The main way an intrusion is executed in a vehicle is by sending a package with a specific ID and a specific data load, as shown in [6]. For this reason, the focus of this thesis will be limited to analysing the ID and data loads of each message in the flow of CAN data.

2.2 One-Class Support Vector Machine

The One-Class Support Vector Machine (OCSVM) [21] is a machine learning algorithm that is used for classification tasks where the motive is to classify each sample as either in or outside a set using only one class as training data. Since OCSVM

is an extension and closely related to the normal Support Vector Machines (SVM) algorithm, a description of SVM will first be presented.

Denote a pair of samples in a data set as $(\mathbf{x}_i, y_i), i = 1, \ldots n$, where $y_i = \pm 1$. Let d be the number of features in each $\mathbf{x}_i$. Then, $\mathbf{x}_i \in \mathbb{R}^d$. In the context of this thesis, $y_i = 1$ indicates a sample being classified as "normal" and $y_i = -1$ as an "attack".

The purpose of the SVM is to create a hyperplane as a classification tool and should be constructed such that it is halfway between the points in opposite classes closest to each other. This main hyperplane can be expressed as $\mathbf{w}^T\mathbf{x_i} - b = 0$, where $\mathbf{w} \in \mathbb{R}^d$ is the normal vector orthogonal to the hyperplane and $b \in \mathbb{R}$ is the scalar offset term.

The separation can be done in the form of a hard-margin using two other hyperplanes; any point having a value $\mathbf{w}^T\mathbf{x_i} - b = 1$ or larger (hyperplane 1) is labelled as normal, while a point having a value $\mathbf{w}^T\mathbf{x_i} - b = -1$ or lower (hyperplane 2) is labelled as an attack. Using this construction, the width of the margin, that is the distance between hyperplane 1 and hyperplane 2 is $\frac{2}{||\mathbf{w}||}$, see Figure 2.1. Maximizing the width is the same as minimizing $||\mathbf{w}||$.

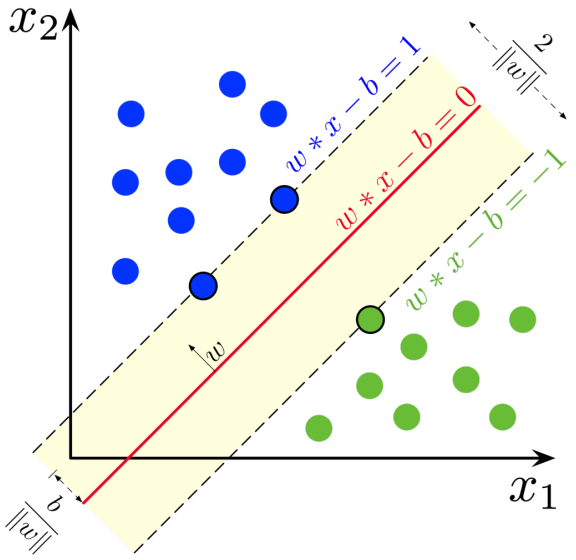


Figure 2.1: Figure showing the margin, the main hyperplane and the two separating hyperplanes for two classes.

Source: [24]

The two separating hyperplanes 1 and 2 can be combined with the expression $y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1$. This leads to the optimization problem:

$$\begin{aligned} &\text{minimize} \quad ||\mathbf{w}||_2^2 \\ &\text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1. \end{aligned} \tag{2.1}$$

The requirement of a hard-margin can be loosened by allowing some points to be in the wrong side of the margin. This is done by instead considering the optimization problem:

$$\begin{aligned} &\text{minimize} \quad \frac{1}{\nu}\sum_i \zeta_i + ||\mathbf{w}||_2^2 \\ &\text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1 - \zeta_i \\ &\qquad\qquad \zeta_i \geq 0 \end{aligned} \tag{2.2}$$

where $\zeta_i = \frac{1}{n}\sum_i \max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i - b))$. The parameter $\nu \in (0, 1]$ in (2.2) controls the trade-off between increasing the margin and assuring that $\mathbf{x}_i$ lies on the correct side of the margin. Solving this optimization problem using Lagrangian duality with Lagrange multipliers α_i leads to the weights $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$. The offset b can be obtained by getting an $\mathbf{x}_i$ on the boundary of the margin and solving $b = \mathbf{w}^T\mathbf{x}_i - y_i$. Only points sitting on the boundary of the margin have $\alpha_i \neq 0$ and serve as support vectors for the separation, hence the term Support Vector Machine. The decision function for determining whether a new point $\mathbf{x}_{new}$ is in one or the other set is:

$$f(\mathbf{x}_{new}) = \text{sign}\left(\sum_i \alpha_i y_i (\mathbf{x}_i \bullet \mathbf{x}_{new})\right). \tag{2.3}$$

The algorithm assumes that the two classes, normal and attack, are linearly separable. To be able to handle non-linearly separable data sets, the Kernel-Trick [25] is used. The decision function in (2.3) as well as the Lagrange dual formulation of the optimization problem in (2.2) depend on the dot product $\mathbf{x_i} \bullet \mathbf{x_j}$, $i, j = 1, \ldots .n$. This can be exploited to transform the pairs of data points to a k dimensional space where they are linearly separable by using kernel functions of the form $K(\mathbf{x_i}, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \bullet \Phi(\mathbf{x}_j)$, $\mathbb{R}^d \to \mathbb{R}^k$, see Figure 2.2. The decision function is then:

$$f(\mathbf{x}_{new}) = \text{sign}(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_{new}) - b) \tag{2.4}$$

An example of a kernel is the radial basis function $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma||\mathbf{x}_i - \mathbf{x}_j||_2^2\right)$, where $\gamma > 0$ is a hyperparameter that can be adjusted.

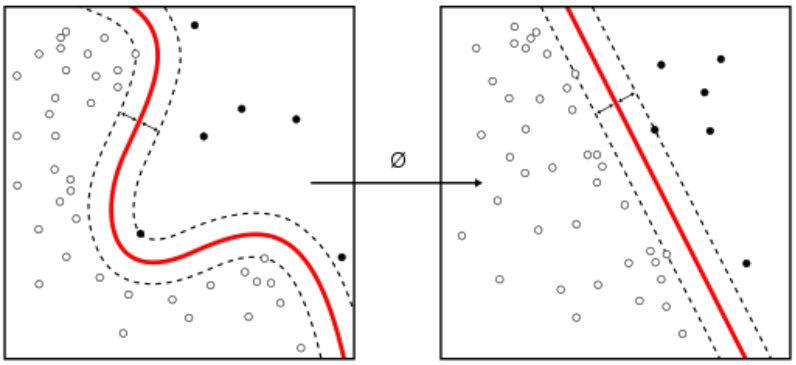


Figure 2.2: Showcase of the usage of Kernels, where a non-linearly separable data set is transformed into a linearly separable one through a kernel function $\varnothing(\mathbf{x})$.
Source: [26]

The One-Class SVM is closely related to the SVM. Just as the SVM, the OCSVM algorithm transforms the datapoints $\mathbf{x}_i$ to a feature space $\mathbb{R}^k$ using the kernel function but now tries to maximize the margin between a hyperplane that the datapoints lie in and the origin. Only one of the two classes is used to train the algorithm. The optimization problem now takes the form:

$$\text{minimize} \quad \frac{1}{\nu}\sum_i \zeta_i + \frac{1}{2}||\mathbf{w}||_2^2 - b \tag{2.5}$$

$$\text{s.t.} \quad (\mathbf{w} \bullet \Phi(\mathbf{x}_i)) \geq b - \zeta_i \tag{2.6}$$

$$\zeta_i \geq 0 \tag{2.7}$$

leading to almost the same decision function as the SVM:

$$f(\mathbf{x}_{new}) = \sum_i \alpha_i K(\mathbf{x}_i, \mathbf{x}_{new}) - b. \tag{2.8}$$

For more information regarding the OCSVM algorithm, see the original paper from B. Schölkopf et al. [21].

2.3 Neural networks

A neural network is composed of neurons, each neuron in its essense being a computational unit. Each neuron takes one or several inputs, processes them according to some rule and gives an output. The rule usually encompasses summing the inputs, adding a bias term and then applying a non-linear activation function. One layer of a neural network can have a varying number of neurons. A neural network typically has multiple such layers, which are connected by having the output of a neuron in one layer connect to the input of one or more neurons in another layer. In a fully-connected feedforward neural network, all the outputs of the neurons in one layer are connected to the input of all neurons in the next layer. A schematic representation of a simple neural network is shown in Figure 2.3.

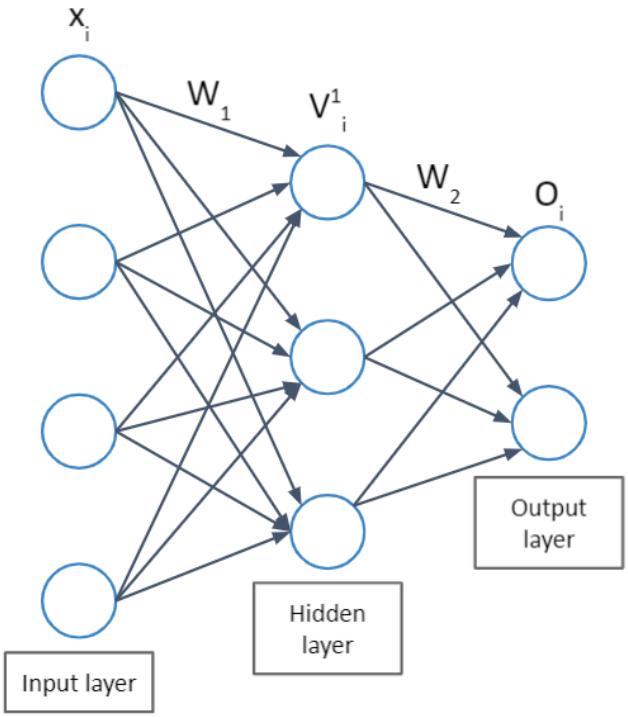


Figure 2.3: Figure of a simple fully-connected feedforward neural network where the nodes represent neurons and the edges represent the connections between neurons. In this figure, there is one input layer, one hidden layer and an output layer.

Let p-dimensional input vectors $\mathbf{x}_i$ with corresponding q-dimensional target vectors $\mathbf{t}_i$, $i = 1, \ldots, n$ form a data set. Denote the state of the layers $l = 0, \ldots, L$ in a neural network as $\mathbf{V}_i^l$. The dimension of each $\mathbf{V}_i^l$ is determined by the number of neurons in that layer. Let the 0th layer be the input to the neural network, that is $\mathbf{V}_i^0 = \mathbf{x}_i$. For each input $i = 1, \ldots, n$, the state of the next layer in the neural network is given by successively calculating:

$$\mathbf{V}_i^l = g(\mathbf{W}^l \mathbf{V}_i^{l-1} + \mathbf{b}^l) \tag{2.9}$$

for each layer l in the network. The weights $\mathbf{W}^l \in \mathbb{R}^{r \times s}$ are the connections between consecutive layers. The dimensions r and s are thus the number neurons in the previous and next layer respectively. The $\mathbf{b}^l \in \mathbb{R}^s$ is the bias terms in each layer. The activation function $g(x)$ is computed element-wise and is usually a non-linear function to create non-linearity between the linear argument and the output of the function. Examples of $g(x)$ are the sigmoid $g(x) = \sigma(x) = \dfrac{1}{1+e^{-x}}$, the hyperbolic tangent $g(x) = tanh(x)$ and rectified linear unit $g(x) = ReLU(x) = \max(0, x)$.

The last layer L of the network $\mathbf{V}_i^L$ is the output of the neural network and is denoted as $\mathbf{O}_i$. The goal of the neural network is to match this output to the target vector $\mathbf{t}_i$. An example of a loss function that is used to measure the discrepancy between the output of the neural network and the target is the mean-squared error (MSE) loss:

$$H = \frac{1}{2} \sum_i ||\mathbf{O}_i - \mathbf{t}_i||_2^2 \tag{2.10}$$

where $(||\quad||_2)$ is the l^2 *norm* of the argument. Clearly, the lower the MSE the lower the discrepancy between the output and the target. Any initial weights $\mathbf{W}^l$ are with a large certainty ill-suited to produce the targets. Because of this, the weights are updated iteratively (after inputs $\mathbf{x}_i$ has gone through the network) using the rule

$$W \longleftarrow \delta W + W \tag{2.11}$$

where the increments δW are given by

$$\delta W = -\eta \frac{\partial H}{\partial w_{ij}}. \tag{2.12}$$

w_{ij} in (2.12) is the i, jth weight element in the matrix $\mathbf{W}^l$ and η is the learning rate hyperparameter, where a larger learning rate corresponds to a larger response relative to the error and hence a larger change in δW. This method of updating the weights iteratively is called *backpropagation through gradient descent*. Feeding all data points $(\mathbf{x}_i, \mathbf{t}_i)$ to the network and backpropagating using the loss function is called batch training. Normally a subset of m samples called minibatches are used for training and backpropagating. One epoch corresponds to when the entire data set has been used in the iterative training process.

For more details on neural networks, backpropagation and speeding up the process of gradient descent (e.g. Adam Optimizer), see [27].

2.4 Autoencoder

An autoencoder [28] is a type of neural network that is used for dimension reduction and anomaly detection. The autoencoder can be divided into two parts. The first part is called an encoder that encodes the input data. Denote a vector input to the autoencoder as $\mathbf{x}$, $\mathbf{x} \in \mathbb{R}^d$. Then, the encoder can be seen as a neural network with the output given by $\mathbf{z} = E(\mathbf{x})$, $\mathbf{z} \in \mathbb{R}^p$. The encoded space is referred to as the latent space. The dimensionality reduction is achieved by having the dimension of the latent space being smaller than the input dimension, that is $p < d$. The dimensionality reduction imposed by the encoder can be used as a tool for visualizing high dimensional data but can also be used as a means to extract important information from the input data.

The second part of the autoencoder is the decoder. The decoder is also a neural network that has as input the output of the encoder and outputs $\mathbf{o} = D(\mathbf{z})$, $\mathbf{o} \in \mathbb{R}^d$.

The goal of the autoencoder is to have the input match the output, that is $\mathbf{o} = D(\mathbf{z}) = D(E(\mathbf{x})) = \mathbf{x}$. The discrepancy between $\mathbf{o}$ and $\mathbf{x}$ is used as a measure of determining the performance of the autoencoder. The behaviour of normal input data is learned when training the autoencoder on the normal data.

2.5 Recurrent Neural Networks

A major limitation with a standard neural network is that it cannot capture time dependencies, since there are no connections between different time steps [27]. To do

this, another type of network is needed, namely a Recurrent Neural Network (RNN) [27]. The basis of the RNN is that it gets as input information from previous time steps, as well as the current time step. The information from previous time steps is often denoted the hidden state of the neuron [27]. In Figure 2.4, the simplest kind of recurrent neural network is shown, with only one hidden neuron. In a timestep t, the state $V(t)$ of this neuron will be dependent on its state in the previous time step, $V(t-1)$, as

$$V(t) = g(w^H V(t-1) + w^I x(t) - \theta^H) \tag{2.13}$$

where g is an activation function, w^H is the weight for the hidden state, $x(t)$ is the input, w^I is the weight between the input and the hidden neuron, and θ^H is the threshold for the neuron. The output O from the neuron is then calculated as

$$O(t) = g(w^O V(t) - \theta^O) \tag{2.14}$$

where w^O is the weight between the neuron and the output, and θ^O is the threshold for the output [27].

The RNN can have much more complex architectures as well, with many neurons and temporal connections between them.

The temporal connections allow the network to take into account or "remember" information from previous times and use this to determine the output of the current time.

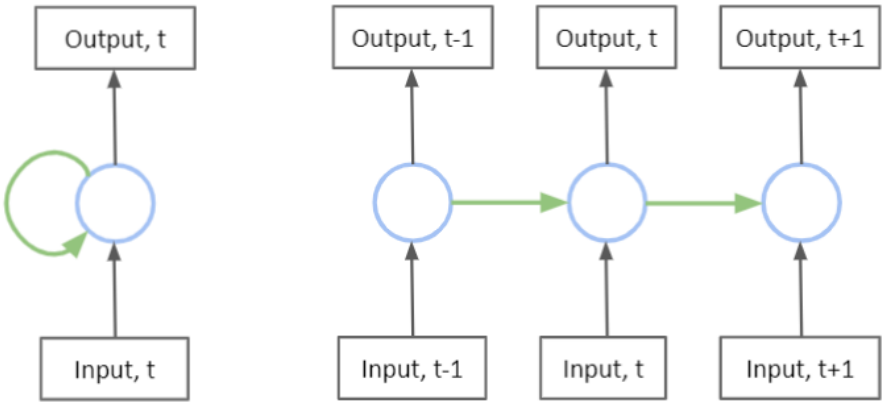


Figure 2.4: Two representations of the same, single neuron RNN. The green arrow represents the hidden state being passed on through time. The leftmost part of the image shows how the hidden state is looped back to the neuron, and the right shows the same neuron unfolded in time.

The network is trained using backpropagation through time, which works similarly to regular backpropagation mentioned previously. The difference is that the gradients are propagated backwards through previous time steps, as well as through the layers. RNNs are even more sensitive to the vanishing or exploding gradient problem than other NNs. This problem arises from the fact that in backpropagation, the gradients are multiplied with each other, and gradients less than 1 lead to smaller and smaller updates, the further back in the network the backpropagation

goes [27]. Since an RNN uses the same weights for a given neuron in each time step, this means that the network fails to store information from many time steps ago, since these gradients become minuscule. To solve this, more advanced types of neurons can be used in an RNN, most commonly Long Short Term Memory (LSTM) units.

2.6 Long Short Term Memory

LSTM was first invented in 1997 by Hochreiter and Schmidhuber [29]. The advantage of using LSTM units in an RNN is that the LSTM unit can regulate what and how much information from previous time steps will be let through. This means that the LSTM is trained to decide what information to remember, and what information to forget, and the network can thus keep important information even from many time steps ago.

In addition to the hidden state being passed along through time and used to update the recurrent neuron, as in standard RNN, LSTMs also have an internal cell state. The cell state gets updated and passed along through time, just like the hidden state. The LSTM unit has four internal layers with trainable parameters, where the calculation of the new cell state and hidden state takes place each time step.

Figure 2.5 illustrates an LSTM cell. The four internal layers are shown as circles, with their respective activation functions (sigmoid and tanh) written out.

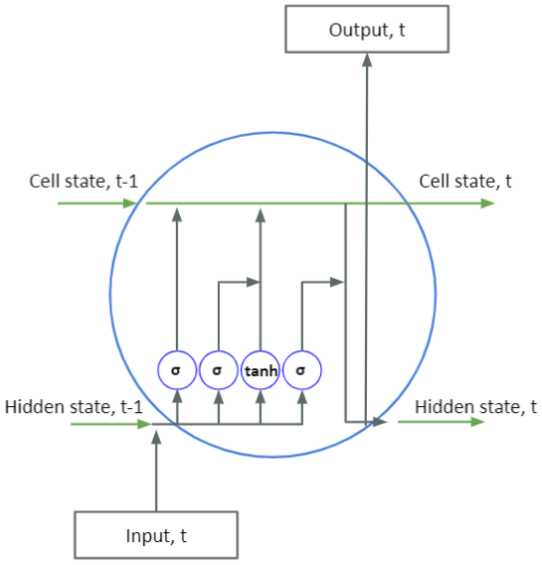


Figure 2.5: An illustration of an LSTM cell. The green arrows represent the information that is being passed through time, in the form of the cell state and the hidden state.

2.7 Convolutional Neural Networks

A Convolutional Neural Network, CNN, is an architecture that, as opposed to a standard neural net, captures spatial correlations from the input data [27]. Because of this property, it is extensively used for image recognition [30], [31], [32], and the three dimensional variant is also used for video analysis [33], [34], [35], as it captures time correlations as well as spatial correlations.

The way CNN works is by applying a set number of filters, which are matrices filled with trainable weights, that slide over the image and take the dot product of the filter with sections of the input. The resulting value of each section is stored in an output matrix. How much the filter moves over the image after each dot product operation is determined by the stride parameter [27]. An illustration of the convolutional process is shown in Figure 2.6 where the stride is 2.

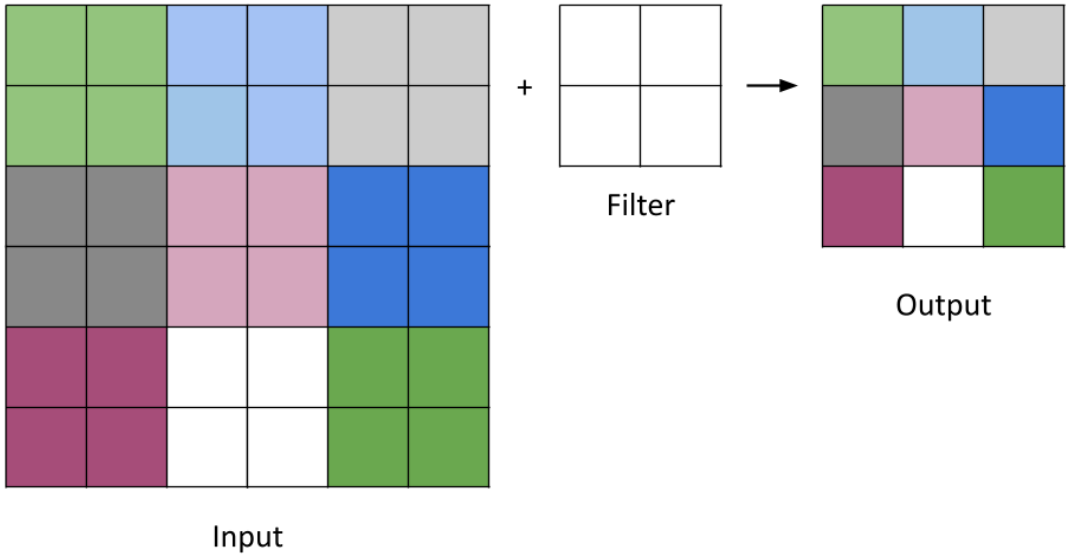


Figure 2.6: Illustration of applying a convolutional filter to a 2-dimensional input. The output from the filter is shown through corresponding colors in the input.

In the 3-dimensional case, the filters are composed of cubes and can slide in all three dimensions of the input.

Usually, after applying a filter to the input, pooling will be performed to decrease the dimension further. The pooling filter will slide over the data similarly to the convolution filter, and give one output per section it covers. The output from the pooling is decided by what type of pooling filter is used, and there are therefore no trainable parameters in this step. For example, if max pooling is used, the maximum value of the section is taken to be the pooling output, and if average pooling is used, the average of the section is the output.

Like in other neural network architectures, an activation function is often applied to the data in each layer. An activation function often used in CNNs is ReLU, Rectified Linear Unit, mentioned in Chapter 2.3.

2.8 Transposed Convolution

Transposed Convolution [36] can be viewed as the opposite of regular convolution, as the dimensions of the input are increased instead of decreased while convolutional operations are performed. Similarly to normal convolution, a filter is applied to the input. Now, instead of taking the dot product, each element of the input is multiplied with each element of the filter, and the output is stored in a new matrix. Thus, for each input element an output of the same size as the applied filter is created [37]. An illustration of this is shown in Figure 2.7.

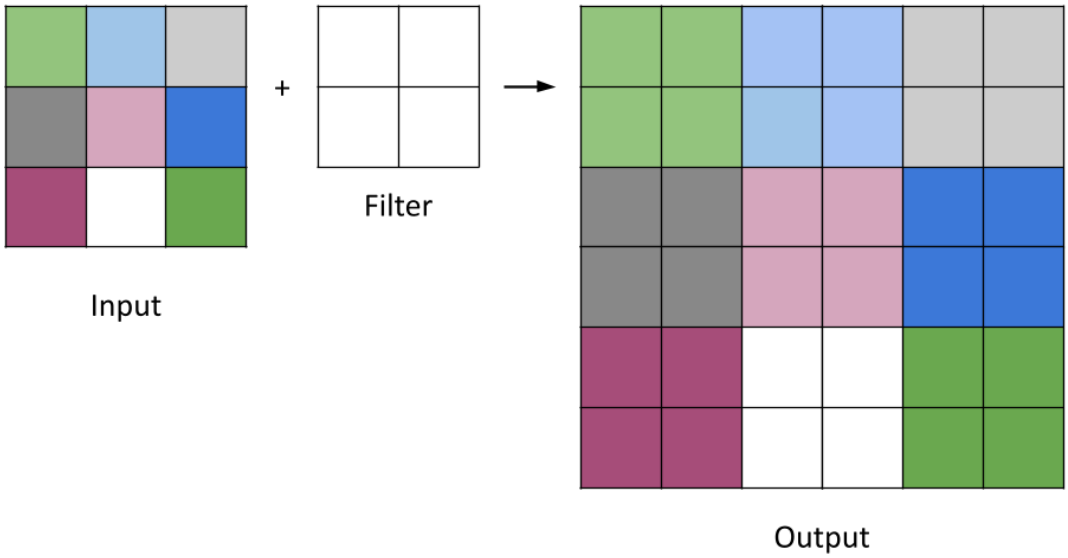


Figure 2.7: Illustration of applying a transposed convolutional filter to a 2-dimensional input. The output from the filter is shown through corresponding colors in the input.

Transposed convolution is often used in the decoding part of autoencoders [38], [39], as a way to increase the dimension back to the original input dimension in a way that incorporates trainable parameters.

2.9 Convolutional LSTM

Convolutional LSTM was first presented in [40] and uses convolutional operations within the LSTM unit. This allows the network to capture both spatial and temporal correlations. The convolutional LSTM, denoted by the inventors as ConvLSTM, takes as input 3-dimensional data instead of 1-dimensional data as in regular LSTM. To handle the extra dimensions in the LSTM unit, the normal matrix operations are replaced by convolutional operations.

2.10 Bidirectional LSTM

Bidirectional LSTM [41] is a type of LSTM network that trains on both the forward and backward direction of an input sequence. This is done by splitting the network up into one part that deals with the reversed sequence, and one that deals with the forward sequence. By utilizing both past and future information to compute an output, bidirectional LSTM often perform better than regular LSTM on certain tasks [42], [43].

2.11 Evaluation Metrics

This section presents metrics that are used in the thesis. Most of the metrics here can be found in [44]. Whenever there are two classes to predict, depending on whether the prediction was right or not, a confusion matrix can be created, which is shown in Table 2.1.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive TP	False Negative FN
Actual Negative	False Positive FP	True Negative TN

Table 2.1: Confusion matrix.

The True Positive Rate (TPR) is a measure of how many correct positive predictions a classifier has made compared to all actual positive instances and is defined as

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{2.15}$$

The False Positive Rate (FPR) on the other hand is a measure of how many actual negative instances were misclassified as positive and is defined as:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}. \tag{2.16}$$

The True Negative Rate (TNR) is the ratio of the correctly predicted negatives compared to all actual negatives:

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}. \tag{2.17}$$

The Positive Predictive Value (PPV) is defined as the ratio between the correctly predicted positives versus all predicted positives:

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}. \tag{2.18}$$

From these metrics, several other metrics can be calculated. The first is the F_β-measure, which is defined as:

$$F_\beta = (1 + \beta^2)\frac{\mathrm{PPV} \cdot \mathrm{TPR}}{\beta^2 \cdot \mathrm{PPV} + \mathrm{TPR}}, \quad \beta > 0. \tag{2.19}$$

The F_β-measure is defined in such a way that it attaches β times the weight to TPR compared to PPV. A large β thus would lead to each instance of false negative affecting the F_β more compared to a false positive. The converse is true for small β.

The accuracy metric is one of the more common metrics and is defined as the ratio between the correct predictions and the overall data, that is:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}. \tag{2.20}$$

The accuracy metric has a flaw, namely that for unbalanced data sets, the accuracy score is close to one even when classifying every point to one class. The balanced accuracy score [45] is less sensitive to unbalanced data sets and is defined as:

$$\text{balanced accuracy} = \frac{\text{TPR} + \text{TNR}}{2}. \tag{2.21}$$

One important evaluation curve to mention is the Receiver Operating Characteristic (ROC) curve. It is a graphical plot that shows the False Positive Rate (FPR) versus the True Positive Rate (TPR) for varying thresholds. Given a threshold separating the two classes, the model will have varying pairs of FPR and TPR and a curve plotting the pairs can be obtained. The best model is one that has no false positives and 100 % TPR which is the point $(0, 1)$ in the ROC curve. The Area Under Curve (AUC) metric is defined to be the area under the ROC curve. It is an important metric that shows how well a discriminator can perform given a number of different thresholds. The perfect discriminator has an AUC $= 1$ since there exists a threshold which gives a perfect TPR and FPR.

3
Methods

3.1 Data

The data used in this thesis is available online [46]. On the website there are in total 5 different data sets. In this thesis, the two data sets fetched from the website are:

1. Gear data set
    - This data set has attack messages that are injected every 0.001s having ID '043f' and a constant data load which is supposed to alter the vehicles gear information. In order to speed up the process of training the models, only the first million messages were used in the analysis.
2. Attack-free
    - A data set without any injected attack data.

The Gear data set has prevously been used in [16], [19]. For illustration, a small portion of the Attack-Free data set is shown in Figure 3.1. The figure clearly shows the time dependent nature of the CAN bus. The attributes of each message in the data sets are:

- Timestamp (formatted as UNIX TIME).
- The ID of the message.
- DLC, integer indicating the number of pairs of hexadecimal numbers in the data field. The DLC can be 2, 5 or 8. Most messages have DLC=8.
- Data load, where each data message data[i], i = 0, ..., DLC contains the data in the data field, given in hexadecimal form.
- Attack, whether the message is injected or part of the normal data. The Attack-free data set does not have this attribute.

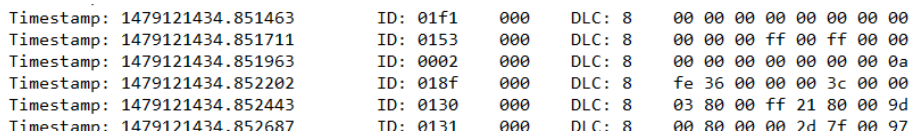


Figure 3.1: A small screenshot of the Attack-Free data set.

3.1.1 The original data

The Gear data set was obtained from a car being turned on and standing still for 30 to 40 minutes. The data was sniffed using the OBD-II port. During the acquisition of the data, attack messages were injected through an ECU. The Attack-free data set was obtained similarly from another vehicle by standing still for around 10 minutes

and without injecting any attack messages. The total number of messages and the IDs in the data sets (after only using a portion of the Gear data set) are shown in Table 3.1.

Data set	Normal Messages	Attack Messages	IDs
Gear	826793	173207	0140 02c0 0350 0370 043f 0440 0316 018f 0002 0153 0260 0130 0131 02a0 0329 0545 04f0 0430 04b1 01f1 05f0 00a0 00a1 0690 05a0 05a2
Attack-free	988871	-	0350 02c0 0430 04b1 01f1 0153 0002 018f 0130 0131 0140 0260 02a0 0316 0329 0545 02b0 043f 0370 0440 04f0 05f0 05a0 05a2 0690 00a0 00a1

Table 3.1: Table showing the number of normal and attack messages together with a list of IDs in the data sets.

3.1.2 Preprocessing of data

As with all data, some preprocessing of the data was done for the different algorithms to work.

- CAN messages having less than 8 pairs of hexadecimal data loads (having DLC < 8) were converted to having DLC $= 8$ by setting all the pairs above DLC to a constant '00'. For example, if the CAN message had DLC=2 and a data load '1A B4', the data load of this message was converted to '1A B4 00 00 00 00 00 00'.
- The data load entries were then converted from hexadecimal to binary data. For example, a data load being '0A B1, ..., 29' is converted to its binary representation '00001010 10110001, ..., 00101001'. This resulted in each message having a 64 bit data load.
- To each message, a feature δ_p is added. This feature is the time difference between the previous and current message. The distribution of δ_p for each data set is given in Figure 3.2.

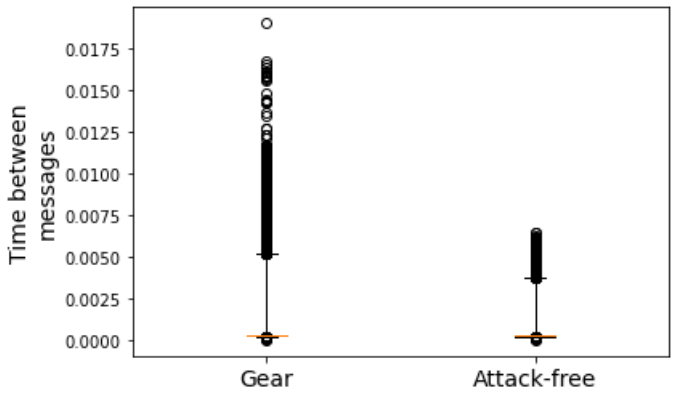


Figure 3.2: A boxplot of the time between messages δ_p.

3.1.3 New data set

Attack messages were simulated using the entire Attack-Free data set. Several problems arose when trying to inject attack messages. The first is *what* message should be injected. The other parameter is the *insertion time window*, when should the messages be injected and for how long? The last parameter is the *insertion frequency*, that is, how often should the data be injected with an attack during the insertion time window.

Firstly, *what* attack message should be injected? The messages should test the detectors on different types of anomalies and hence include a variety of message data loads and IDs. Some data loads in each message should be injected such that they have been seen before in the flow of CAN messages and some that have not. Attack messages which have a data load that is the same as the message data loads seen in the normal data would only be anomalous with respect to time, while messages that have a data load differing from the normal data would then be anomalous with respect to time and data load content. This tests the ability for the detectors to detect different type of anomalies. With this in mind, Algorithm 1 is used to create attack messages to be inserted in the Attack-free data set. For each ID, the algorithm takes all the messages in the Attack-free data set having that ID. Then, the mean of each bit in the 64 bit data loads of these messages are calculated. If the mean of the bit is equal to or above 0.5, the bit is set to 1, else it is set to 0. Thus, a 64 bit data load containing zeros and ones is created for each ID.

Algorithm 1: Algorithm to create messages to be injected as attack.

Data: The Attack-free data which contains messages having 64 bit data loads and one out of 27 unique IDs.
Result: A **message matrix** containing for each of the 27 IDs the 64 bit data load to be inserted as attack data load.

```
/* Create message matrix ⟵ [27 by 64] matrix of zeros    */
for i = 1 to i = 27 do
    current ID data ⟵ all the data messages of ID i;
    for j = 1 to j = 64 do
        column ⟵ current ID data[:, j];
        mean of column ⟵ Mean of column;
        if mean of column ≥ 0.5 then
            message matrix[i, j] ⟵ 1
        else
            message matrix[i, j] ⟵ 0
        end
    end
end
```

How common is it for each ID to have a data load created by the algorithm to contain exactly the same data load in the normal data? To answer the question, the ratio of data loads for each ID in the Attack-free data set that have the same data loads as the data loads created by the algorithm is calculated. The result is

given in Table 3.2. As seen from the table, depending on the ID, messages that will be inserted will be exactly the same, some of them will be the same or none of them will be the same compared to the ones encountered previously in the normal CAN data.

ID	**0350**	**02c0**	**0430**	**04b1**	**01f1**	**0153**	**0002**	**018f**	**0130**
Ratio	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.006	0.0
ID	**0131**	**0140**	**0260**	**02a0**	**0316**	**0329**	**0545**	**02b0**	**043f**
Ratio	0.0	0.0	0.0	0.047	0.0	0.028	0.289	0.001	0.0
ID	**0370**	**0440**	**04f0**	**05f0**	**05a0**	**05a2**	**0690**	**00a0**	**00a1**
Ratio	0.265	0.0	0.0	1.0	1.0	0.0	0.668	0.0	0.025

Table 3.2: Table showing the ratios of the data loads that are the same when comparing all the the 64 bit data loads for an ID to the data loads created by using Algorithm 1 for the same ID. A ratio of 0.0 for an ID means that none of the data loads created by the algorithm for that ID are the same in the normal data.

The *insertion time window* is chosen such that a burst of attack messages with a specific ID are inserted at specific times while at the same time retaining a large ratio between normal and attack data. The IDs of the messages to be inserted follow the order of the Attack-free data set shown in Table 3.1, beginning in the top-left and continuing to the right. In the first time window, the 64 bit message corresponding to ID '0350', which is found in the first row of the **message matrix** in Algorithm 1 is inserted at a specific insertion frequency. This is done for all IDs twice. The time between windows is set to 4s. Each window is 0.2s long. A summary of the details of the insertion time window is shown in Table 3.3.

ID	0350	02c0	...	00a1	0350	0545	...	00a1
Start Time (s)	260	264	...	364	368	372	...	472
End Time (s)	260.2	264.2	...	364.2	368.2	372.2	...	472.2

Table 3.3: Details of insertion time windows.

Regarding the *insertion frequency*, the following is argued. The lower an injection frequency is, the harder it is for any detector using a window to correctly classify an anomaly. To put it another way, it is harder to detect a window of say 40 messages as anomalous if only one of the messages in the window is an attack compared to when 30 of them are attacks. As an overview, the median time between each message for each ID is given in Figure 3.3.

The median time difference between each message is around 0.01s for most of the IDs. Thus, an insertion frequency of 0.04s is chosen as a balance between having a lower frequency than the normal messages but having enough attack data messages to make inferences on.

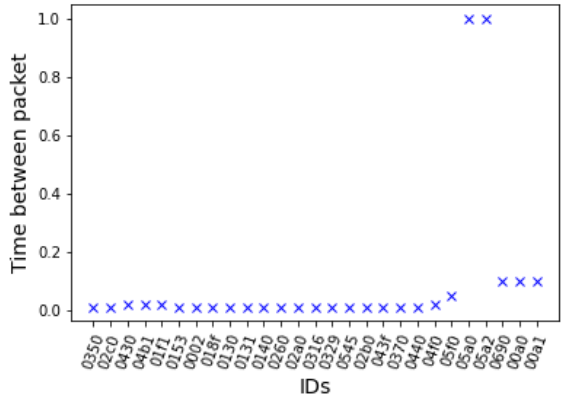


Figure 3.3: The median time between messages for each ID in the Attack-free data set.

3.1.4 Separation of data

In order to test the generality of the models, the data was divided into training, validation and test subsets. Figure 3.4 shows the divisions and the labels of the Attack-free and Gear data sets.

Out of the 988871 messages in the Attack-free data set, 50% were split for training and the other 50% were split for validation and testing purposes. From the validation and testing messages, 70% were split to use for validation containing only normal messages. The attack messages from the previous chapter were injected into the validation and test portion of the Attack-free data set. Henceforth, the entire Attack-Free data set together with the inserted attack messages is referred to as the "New data set".

A similar approach was done for the Gear data set. Since attack-messages were injected continuously, the entire data set is a mixture of both normal and attack data. The first 40% of the data set was split for training, the next 35% for validation and the last 25% for testing. Whenever needed, only normal messages from the training and validation subsets were used, which are labeled as "Training Normal" and "Validation Normal" in the Gear data set portion of Figure 3.4.

For the different upcoming methods of anomaly detection, the messages in the subsets were divided into windows. A number of $L = 40$ messages were taken at a time in the same order as they appear in the subsets. If one of the messages in the window had an attack, the whole window was labeled an attack. The number of attack and normal windows in each subset is shown in Table 3.4.

3.2 Evaluating Networks

In this chapter, a list of the different metrics used to evaluate and design the models will be given. Positive (+1) and negative (-1) labels are given to attack and normal windows respectively.

In this thesis, the metrics to evaluate the networks on are chosen to be the TPR, FPR, balanced accuracy and F_β with $\beta = 0.1$ and $\beta = 10$. The TPR is used to

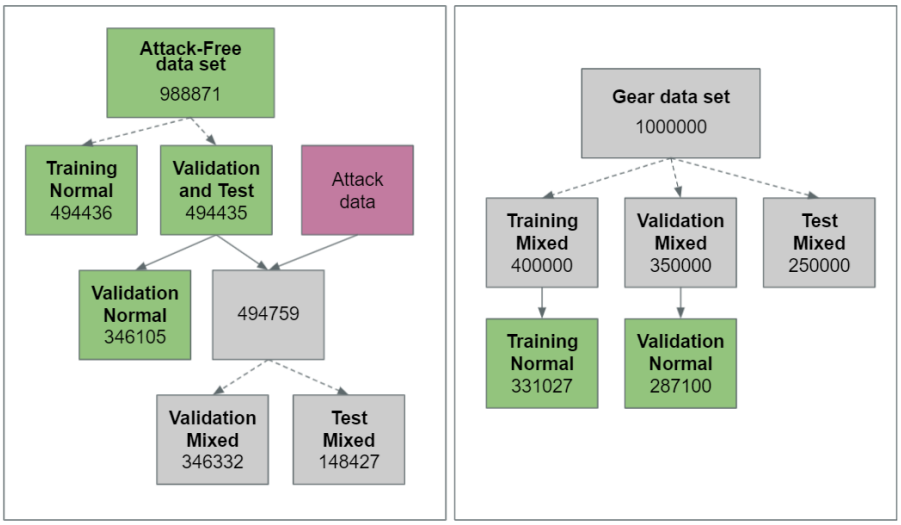


Figure 3.4: Showcase of the subsets created after dividing the Attack-free and Gear data sets. The number of messages in each subset is shown in each box. A green box means that the data has exclusively normal data, while the gray boxes represent mixed normal and attack data.

Subset	Training Normal New	Validation Normal New	Validation Mixed New	Test Mixed New	Training Gear	Validation Gear	Test Gear
Normal Windows	12360	8652	8400	3644	4557	3737	2990
Attack Windows	-	-	258	66	5443	5013	3260

Table 3.4: Table showing the number of normal and attack windows for each subset.

show how well the detectors are able to correctly predict attack windows. The FPR is used since the metric shows how prone the detector is to falsely classify normal windows as attacks. The balanced accuracy is a way of generally seeing how many true attack and normal windows the detectors find. Lastly, the F_β with $\beta = 10$ is used to indicate how good the detectors are when 10 times the importance is given to false negatives. This means that the larger β grants higher weight to attacks being classified as normal. The converse holds for $\beta = 0.1$.

The error measure used in the different neural networks for each sample throughout the thesis is defined as the l_1 *norm*:

$$e_{sample} = ||\mathbf{X} - \hat{\mathbf{X}}||_1 \tag{3.1}$$

where $\mathbf{X}$ is the input sample which can be a vector, matrix or a tensor depending on the input to the neural network, and $\hat{\mathbf{X}}$ is the predicted output sample which has the same dimensions as the input sample. Given a threshold, this error can be used to classify the input as either being an attack or normal. Thus, in connection to this metric and given a threshold, the ROC curve explained in Chapter 2.11 is used to show the performance of the neural network predictions.

Finally, the loss the neural networks use for training is binary cross entropy loss:

$$loss = -\frac{1}{\text{batch size}} \sum_i y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i) \tag{3.2}$$

where i is the index over each element in a batch. y_i and $\hat{y}_i$ are the true and predicted values of the element respectively.

Two ways of visualising the e_{sample} will be done by showing box plots and scatter plots. The orange lines in the box plots represent the median of a distribution and the whiskers represent the 1st and 99th percentile of a distribution. Points below or above the whiskers are shown as black circles.

3.3 Networks

To capture the normal behavior of the CAN bus, several different networks were created and tested. These all built upon the idea that if the network could be trained to reproduce normal data well enough, any successive data that was *not* reproduced well by the network would be deemed an anomaly. The detected anomaly could then be flagged as a potential attack. For the neural networks, the difference between the network considering data to be normal or anomalous would be the value of the reconstruction error. Normal data would give a lower loss compared to anomalous data. To this end, all networks were exclusively trained on normal data, but validated and tested on both anomalous and normal data.

When training the networks, the optimizer used was the Adam Optimizer with the *learning rate* parameter fixed to 0.001. Early stopping was employed to avoid overfitting. This means that if the validation loss did not decrease after a certain number of consecutive epochs, the training was stopped. The number of epochs allowed before stopping the training is determined by the *patience* parameter.

Two different approaches were taken regarding how to design the anomaly detector networks. The first one, Branch 1, combined a neural network with OCSVM. The description of how this was done can be found in Chapter 3.3.1. The second one, Branch 2, used only a neural network. The description of how this was done can be found in Chapter 3.3.2.

For both branches thresholds to distinguish normal data from attack were optimized for each network and data set. This was because the different data sets were quite different in nature since they were obtained from different vehicles, and yielded different results when being reconstructed by the networks, as can be seen in Chapter 4. Furthermore, because of the varying nature of CAN for different vehicles, it is argued that any anomaly detector would first need to be trained on normal data from a specific vehicle before being implemented into it. This would also mean that different thresholds would need to be set for different types of vehicles, since the thresholds are based on the normal data. It is therefore natural in this case to use different thresholds for the different data sets.

All networks were created using Keras [47] and Tensorflow [48] in Python.

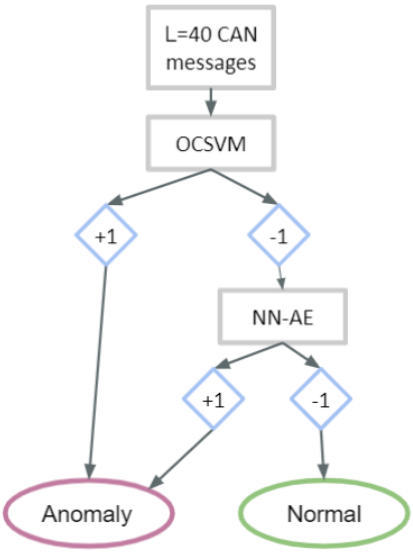


Figure 3.5: Flow diagram showing the flow of Branch 1 method.

3.3.1 Branch 1

Branch 1 of the anomaly detector proposed in this thesis contains two different methods to detect injected messages. In the CAN flow, attacks can happen in several ways. A message is inserted at a different time compared to normal, a message can be inserted at a normal time but the data load of that message is harmful or both the timing and the data load of the message is abnormal. To solve for a message being an anomaly with respect to time the OCSVM algorithm is used. To detect intrusions where only the data load of a message is changed, an Neural Network Autoencoder (NN-AE) is used. A flow diagram showing the overall structure of this branch is shown in Figure 3.5. All models are trained, validated and tested separately on the New and Gear data sets.

3.3.1.1 OCSVM

The input to this branch is a window of $L = 40$ CAN messages which contained the following attributes

- An ID.
- A 64 bit CAN data load.
- δ_p The time difference between the previous message and itself.

The OCSVM algorithm is heavily inspired by [49]. From each window, a vector $\mathbf{v}_O$ with the following features was collected as an input to the OCSVM algorithm

- The sample mean $\mu = \frac{1}{L}\sum_{l=1}^{L}\delta_l$ of all the messages in the window.
- The sample variance $\sigma = \frac{1}{L-1}\sum_{l=1}^{L}(\delta_l - \mu)^2$ of all the messages in the window.

There are thus 2 features used in the OCSVM classifier.

The data sets used for training the OCSVM were the training normal subsets.

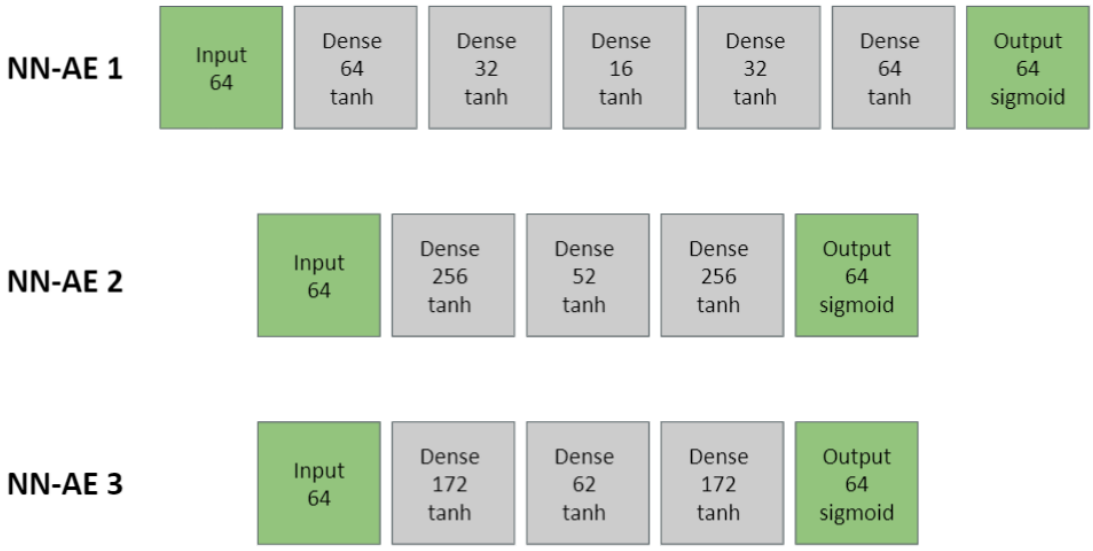


Figure 3.6: Figures showing the architectures of the different autoencoders. Each box in the figure represents a neural network layer. In each box, the type, the number of neurons and the activation function used in a layer is shown.

For the OCSVM algorithm, the radial basis function is used as a kernel to allow non-linearity in the distribution of the training data. Also, a soft margin is used to allow some data points in the OCSVM to be misclassified (leading to higher generalization of the data). The hyperparameters to be adjusted for this algorithm are γ and ν. These hyperparameters were adjusted iteratively such that they maximized the balanced accuracy score in the mixed validation subsets. First, a grid containing many (γ, ν) pairs of values was created. Based on the balanced accuracy score using the hyperparameter values from the grid, the interval for both γ and ν values was adjusted until the score did not increase further.

3.3.1.2 The Neural Network Autoencoder

Autoencoders as noted in Chapter 2.4 can be used as a method for anomaly detection. If the OCSVM deems the window as normal (-1), the window (that is, the 40 messages of a window) is given as input to an autoencoder, which henceforth will be called the Neural Network Autoencoder (NN-AE). The 64 bit data load of these 40 messages are evaluated individually on the NN-AE. Since the data load of a message is 64 bit, the input to the NN-AE is 64 bit. Also, since it is an autoencoder, the output has the same dimensions as the input, that is 64 elements.

$\mathbf{X}$ and $\hat{\mathbf{X}}$ defined in (3.1) are thus 64 dimensional vectors. Since the error scalar e_{sample} is computed message-by-message and a window is of size 40, it leads to there being a total of 40 errors e_{sample}. If any of these 40 errors are above a certain threshold th_N or th_G for New and Gear data sets respectively, the message and thus the entire window is deemed as an anomaly. These thresholds were chosen such that they are larger than any of the errors of normal messages.

The loss used for this neural network is the binary crossentropy loss defined

in (3.2). The NN-AEs are trained on the training subsets containing only normal messages. The validation data when training the NN-AEs are the validation subsets containing normal messages. The different tested architectures compared are given in Figure 3.6. The final architecture that is chosen for the New and Gear data set respectively is the architecture having the lowest validation loss. The data used to set the thresholds th_N and th_G were the mixed validation subsets.

Shared parameters for all the NN-AEs is a batch size of 400, the maximum number of epochs to 400 and the *patience* parameter set to 20.

3.3.1.3 Combining the OCSVM and NN-AE methods

The combination of this branch is straightforward and shown in Figure 3.5. Each window with size $L = 40$ messages is first sent to the OCSVM. If the OCSVM given the window input $\mathbf{v}_O$ outputs +1, the whole window is classified as an anomaly and the final label of the window is (+1). If it outputs (-1), the input window of $L = 40$ messages is sent as input to the NN-AE. Again, this is motivated by the fact that the data loads of the messages in the window can be malicious even though the messages within the window are normal with respect to their order and time of appearance. The data load of each message is fetched and the e_{sample} is calculated. If any of the 40 e_{sample} values are above the threshold th_N or th_G for the New and Gear data set respectively, the window is deemed as an anomaly, that is (+1). Else, the window is deemed to be normal (-1). The output of this branch is then later combined with Branch 2, see Chapter 3.4. The final performance of this branch on the unseen test subsets will be presented in Chapter 4.1.3.

3.3.2 Branch 2

For Branch 2, several different architectures of neural networks were tested, to see which would work best for the task at hand. The aim was to have a network that would recreate the normal CAN input as well as possible, while also *failing* to recreate the anomalous data. For each type, many different experiments were performed, to find the most suitable number of layers, hyperparameter values, activation functions and so forth. The best versions of each type is presented here. The different types presented are LSTM Autoencoders, 2D Convolutional LSTM Autoencoders, 3D Convolutional Autoencoders, and convLSTM Autoencoders. When training the networks, a batch size of 100 was used, and early stopping with *patience* = 10. Details on how the networks were designed are presented below. For each architecture tested, an illustration of the network is shown together with details on parameters used in each layer. Only the parameters values that differed from the Keras [47] default values are shown. Each network was trained, validated and tested separately on the New and the Gear data sets. For the New data set, the networks were trained on the normal training subset, using the normal validation subset as validation during training to avoid overfitting, and then tested on the mixed validation subset. The best performing network on the mixed validation subset was then further tested on the mixed test subset. For the Gear data set, all networks were trained on the training normal subset, using the normal data from the validation subset as validation during training. These were then all tested on the full mixed

validation subset, and one network was chosen to be tested on mixed test subset as well.

3.3.2.1 LSTM Autoencoder

When designing the LSTM Autoencoder (LSTM-AE), inspiration for the architecture was taken from [50] and [51]. Only the CAN data loads where used. The time sequence of 64 bit loads was then divided into time windows of size L time steps, with an overlap between consecutive windows of size m. An illustration of this is shown in Figure 3.7, with a window size of 4 and an overlap of 2. No overlap was used during inference. Like in [50], a window size of $L = 40$ was used. An overlap

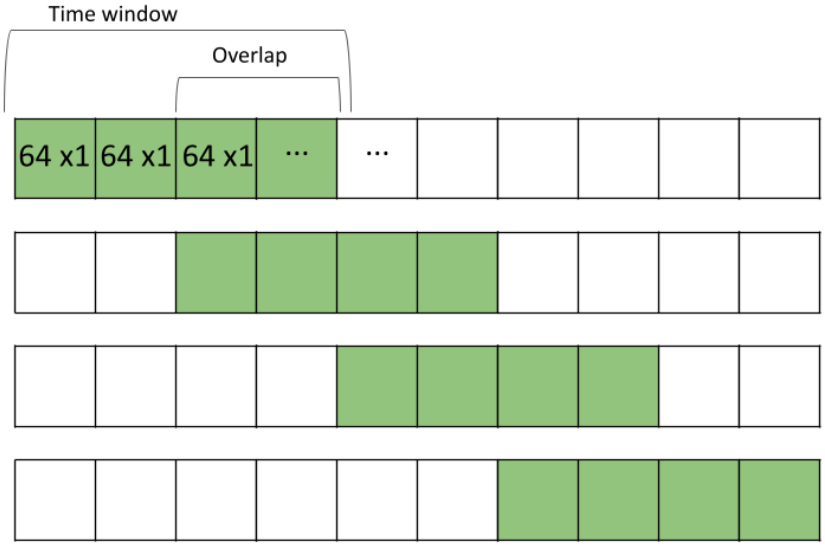


Figure 3.7: A representation of how the data was divided for the LSTM AEs.

of 20 was used to not lose correlations between different windows during training.

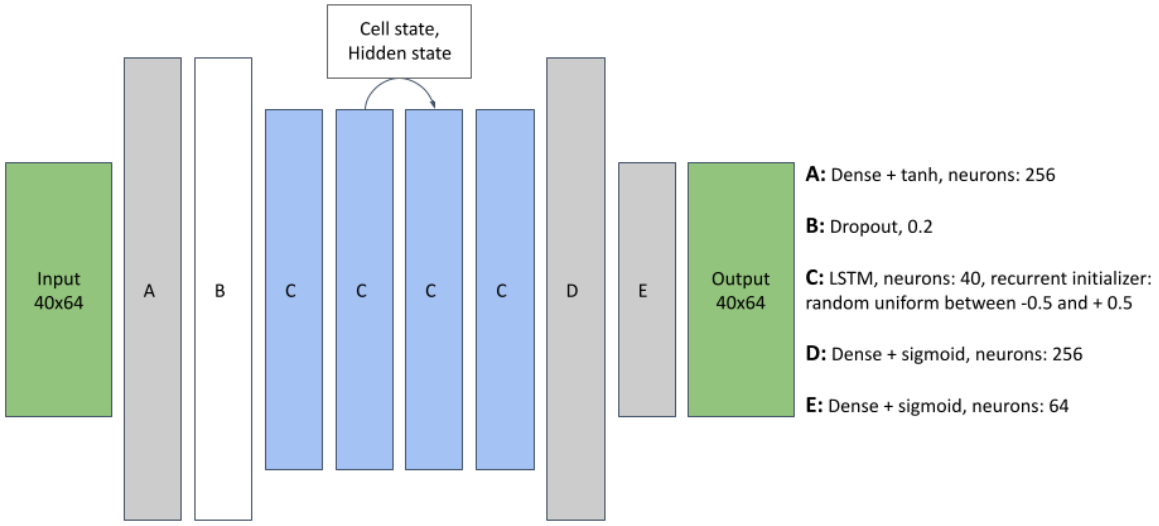


Figure 3.8: The architecture of Large LSTM.

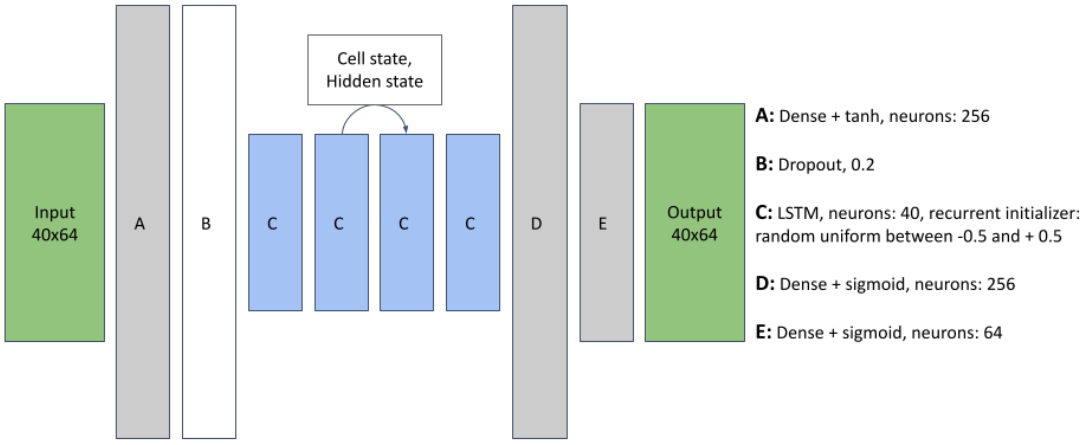


Figure 3.9: The architecture of Small LSTM.

For the LSTM-AE, two different architectures were tested. The first one is denoted Large LSTM and is very similar to the one used in [50]. Its architecture is shown in Figure 3.8. However, this architecture does not press down the input dimensions in the encoder part, and thus loses some of the advantage of a traditional Autoencoder. With this in mind, the network shown in Figure 3.9 was designed, in which there are only 40 LSTM units in each LSTM layer, instead of 128. It thus got

the name Small LSTM. Furthermore, unlike in [50], both networks were trained and tested on all data together, and not separately for each CAN ID. Thus, one model was created per data set used.

3.3.2.2 2D Convolutional LSTM Autoencoder

To capture correlations between IDs and data loads as well, a convolutional LSTM Autoencoder was built. The idea behind this was to capture correlations between IDs and data loads with 2d convolutions, then capture time correlations with LSTM, and then use transposed convolution to get the dimension of the output to be the same as that of the input. The final layer was set as Dense since this was found to improve performance.

To this end, each ID was transformed into 16 bit binary form, as this was found to work better than decimal form. After this, the data was divided into one 2d matrix for each time step, to be able to perform 2d convolutions. Each matrix consisted of 64 columnwise copies of the current 16 bit ID above its 64 bit data loads. This is illustrated in Figure 3.10. Similarly to the LSTM AE data, an overlap in time of 20 messages was used when training the networks.

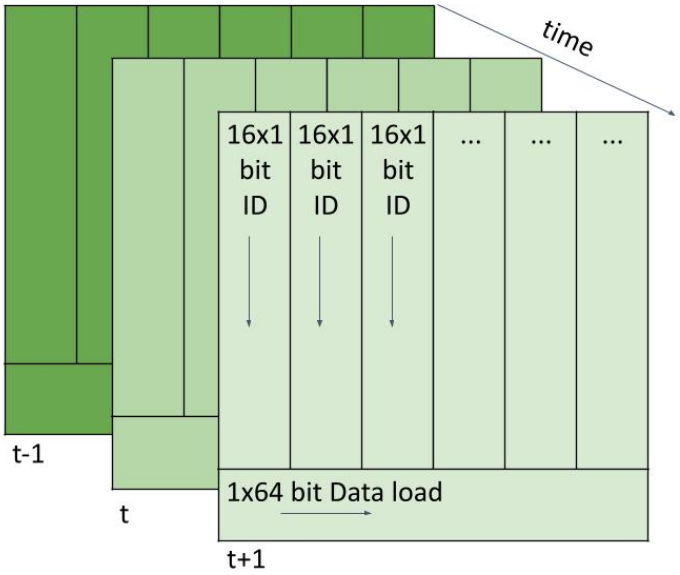


Figure 3.10: A representation of how the data was divided for the 2D CNN LSTM.

Two different configurations of this network were tested. In the first one, illustrated in Figure 3.11, the convolutions were performed by letting each time step be a channel, similarly to how in 2d convolutions over color images, each color is a channel [52]. To match the input dimensions to how they are supposed to be stated in Keras [47] in this case, the input data was transposed to get the dimensions $64 \cdot 17 \cdot 40$. Using this configuration meant that there were separate weights for each channel, but since the filters only slide over the first two dimensions, no correlations were captured between channels. This network is denoted 2D CNN LSTM.

In the second one, the convolutional and transposed convolutional layers as well as the dense layer were put within a Time Distributed wrapper. What this does is it applies the layer within to each time step matrix independently, but using

the same weights [53]. To this end, the input data was transposed and an extra dimension was added, to get the dimensions $40 \cdot 64 \cdot 17 \cdot 1$.The network is illustrated in Figure 3.12 This network is denoted TimeDistributed 2D CNN LSTM.

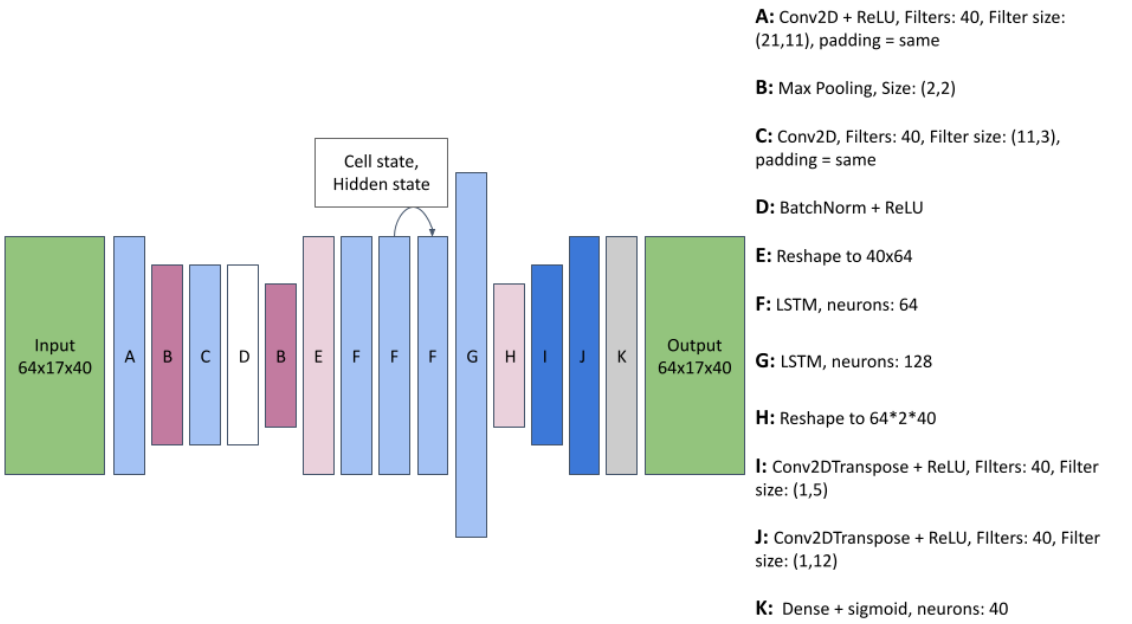


Figure 3.11: The architecture of the 2D CNN LSTM.

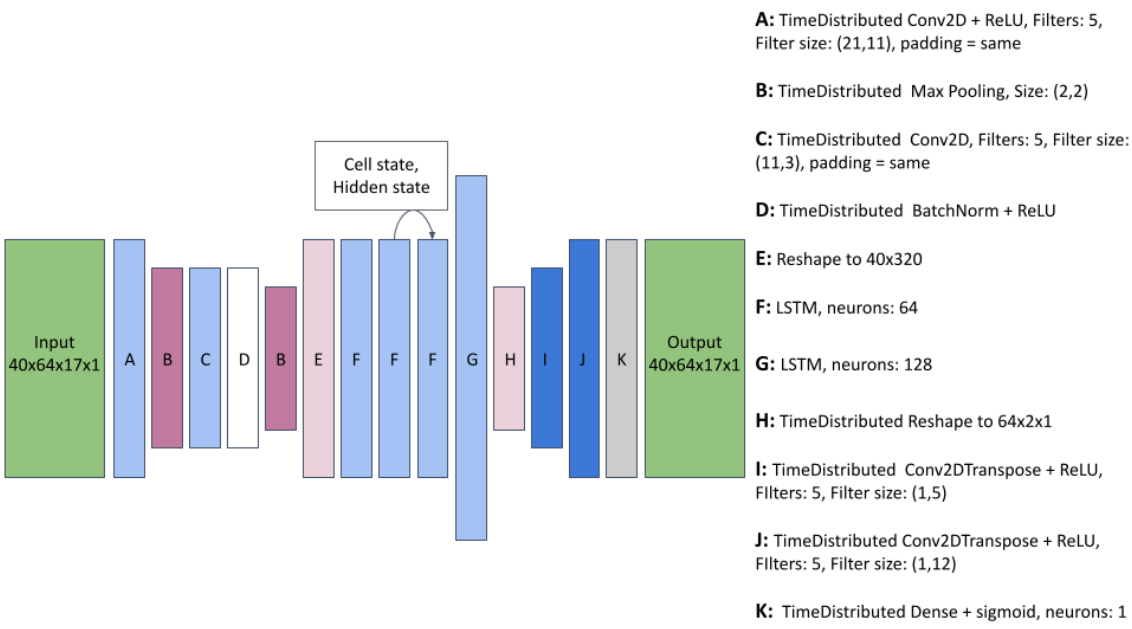


Figure 3.12: The architecture of the TimeDistributed 2D CNN LSTM.

3.3.2.3 3D Convolutional Autoencoder

To test whether it would be possible to capture correlations in the time dimension without using LSTM, a 3D Convolutional Autoencoder was built, denoted 3D CNN.

For this network, the same data preprocessing as is shown in Figure 3.10 was used. However, the time dimension was now included in the convolution, using three dimensional filters. Similarly to the TimeDistributed 2D CNN, the input data was made to have the dimensions $40 \cdot 64 \cdot 17 \cdot 1$. The architecture of the 3D CNN is shown in Figure 3.13.

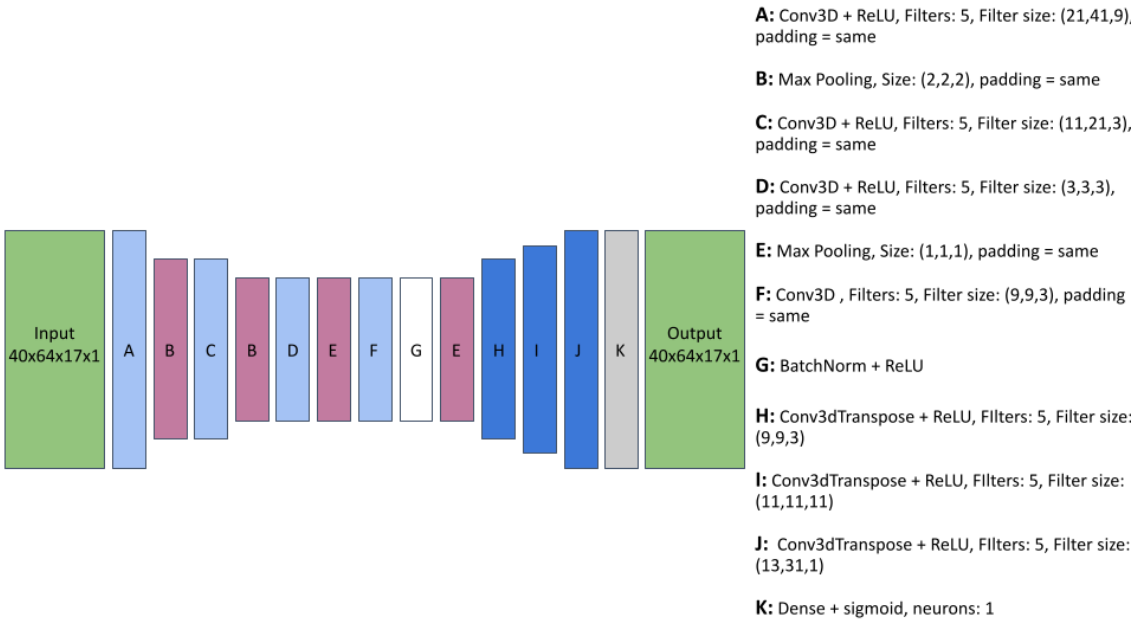


Figure 3.13: The architecture of the 3D CNN

3.3.2.4 Convolutional LSTM Autoencoder

The Convolutional LSTM networks used ConvLSTM units to capture correlations in time, ID, and data loads. Both a unidirectional and a bidirectional version were tested. Both used the same type of input data as the 3D CNN and TimeDistributed 2D CNN, namely with dimensions $40 \cdot 64 \cdot 17 \cdot 1$. The unidirectional network is illustrated in Figure 3.14 and is denoted ConvLSTM and the bidirectional network is illustrated in Figure 3.15. This is denoted Bidirectional ConvLSTM.

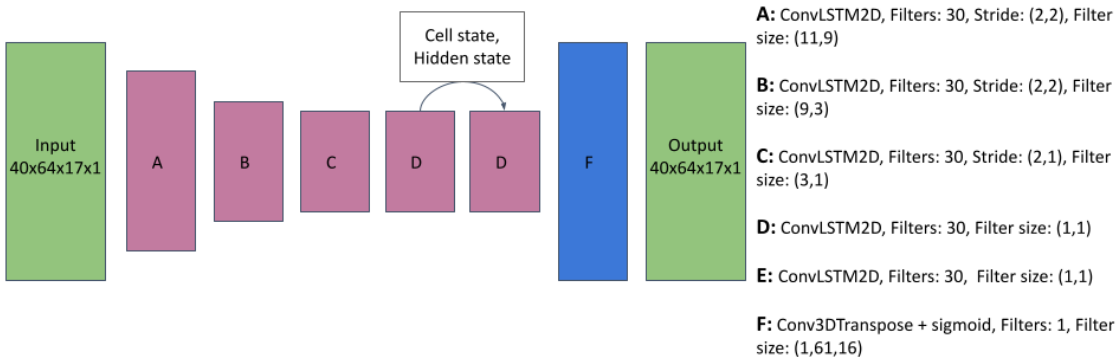


Figure 3.14: The architecture of the ConvLSTM.

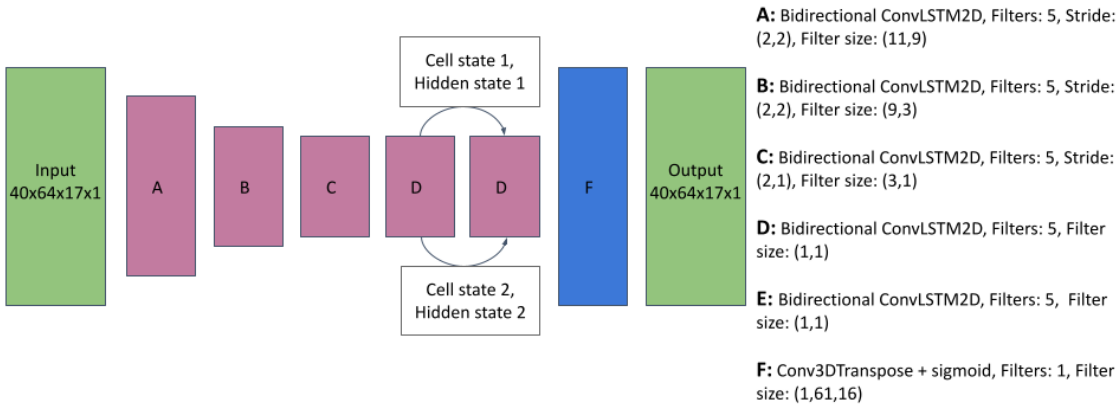


Figure 3.15: The architecture of the Bidirectional ConvLSTM. Since the time sequence is processed both forwards and backwards, there are two cell states and two hidden states in each unit.

3.4 Combined Anomaly detector

For each window of $L = 40$ messages, from both Branch 1 and 2, a prediction on whether the window is normal (-1) or an anomaly (+1) is made. The way these two

branches are combined is by using two logical operators, see Table 3.5. To gauge which operation is best, the two operators will be compared using the different metrics defined in Chapter 3.2. In words it means the AND operation labels a window as an attack only if both Branch 1 & Branch 2 agree that it is an attack whereas the OR operation labels a window an attack whenever either Branch 1 or Branch 2 deems the window as an attack.

Branch 1 Output	**Branch 2 Output**	**AND Combination**	**OR Combination**
-1	-1	-1	-1
-1	1	-1	1
1	-1	-1	1
1	1	1	1

Table 3.5: Table showing the two combinational methods AND and OR to be used.

4
Results

4.1 Branch 1 Results

In this chapter, the final selection of the OCSVM and NN-AE models together with the thresholds th_N and th_G, closely following Chapter 3.3.1, will be motivated and presented. This chapter ends with showing the performance of this Branch on both the New and Gear data sets.

4.1.1 Results of OCSVM

The (γ, ν) pairs that were tested according to the methodology of Chapter 3.3.1.1 and iterated on to achieve the largest possible balanced accuracy score are shown in Table 4.1.

Iteration Number	**ν range**	**γ range**	**Best ν**	**Best γ**	**Best Balanced Accuracy**
New Dataset					
1	0.0001 0.0002 ... 0.01	10 60 100 600 1000 3000 5000 7000 9000, 10000 20000 30000 40000	$4 \cdot 10^{-4}$	9000	0.99994
2	$3 \cdot 10^{-5}$ $4 \cdot 10^{-5}$ ... $5 \cdot 10^{-4}$	8500 8510 ... 9500	$4 \cdot 10^{-4}$	8800 8810 ... 9020	0.99994
Gear Dataset					
1	0.1 0.2 ... 0.9	1000 2000 ... 10000	0.1	1000	0.88561
2	0.05 0.051 ... 0.15	1000 2000 ... 10000	0.066	10000	0.89793
3	0.06 0.061 ... 0.07	8000 9000 ... 24000	0.066	10000 11000 12000	0.89793

Table 4.1: The iterations of hyperparameter tuning of the OCSVM algorithm for both datasets.

The final parameters ($\gamma = 8900$, $\nu = 0.0004$) and ($\gamma = 11000$, $\nu = 0.066$) defining the optimized OCSVM models for the New and Gear data sets respectively were chosen.

The final scores on the mixed validation subsets for the optimized OCSVM models are given in Table 4.2. As the table shows, the OCSVM alone performs very well on the mixed validation subset of the New data set with very low FPR and a perfect TPR score. On the mixed validation subset of the Gear data set, the performance is not as impressive, but it is still good with a TPR larger than 0.83 and an FPR lower than 0.04.

Data set	Accuracy	Balanced Accuracy	TPR	FPR	$F_{\beta=0.1}$	$F_{\beta=10}$
Gear	0.89269	0.89793	0.83768	0.04181	0.95839	0.83874
New	0.99988	0.99994	1.0	0.00012	0.99618	0.99996

Table 4.2: The final metric scores on the mixed validation subsets using only the OCSVM algorithm.

4.1.2 Results of NN-AE

This chapter shows the results using the methodology of Chapter 3.3.1.2 and the final NN-AE architectures chosen to use later on for combining the OCSVM and NN-AE. Running the three different NN-AE architectures shown in Figure 3.6 on the subsets of training and validation containing only normal messages, Table 4.3 was obtained.

	New data set		Gear data set	
	Early Stopping	***Validation Loss***	***Early Stopping***	***Validation Loss***
NN-AE-1	37	8.2×10^{-2}	54	2.2×10^{-4}
NN-AE-2	25	7.8×10^{-4}	35	1.2×10^{-4}
NN-AE-3	400	7.4×10^{-8}	42	2.1×10^{-4}

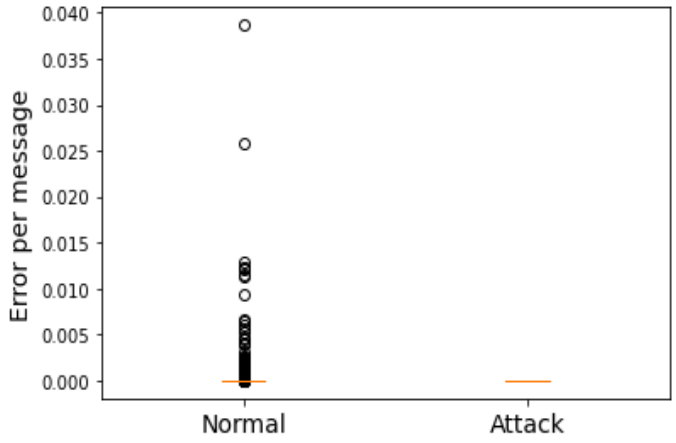
Table 4.3: Table showing the performance of the NN-AE models on the normal validation subsets. The Early Stopping columns show the number of epochs the neural networks ran until the validation loss did not increase in *patience* number of epochs.

Clearly the NN-AE-3 and NN-AE-2 models gave the best validation loss for the New and Gear data sets respectively. These models were chosen for further inference. Notice that stopping was never achieved on the NN-AE-3 for the New data set, that is, this model could have achieved even further validation loss. Since the last layer of the NN-AE-3 is a sigmoid function, each element predicted will, unless it is extremely close to 0, never reach zero. Thus, the result is satisfactory and the the model is said to be trained.

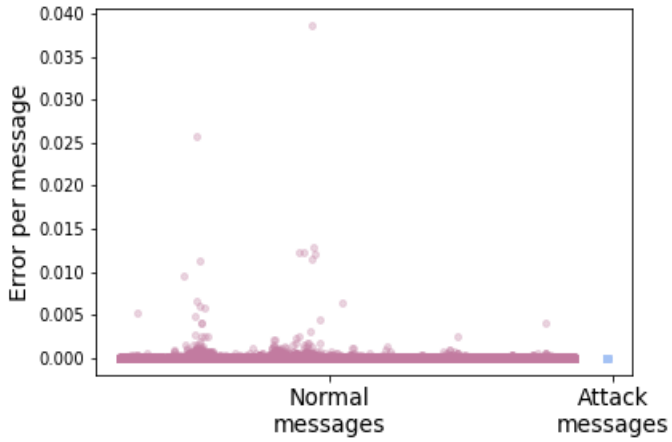
After obtaining the final NN-AE models, each message from the mixed validation subsets, which contained both attack and normal data, were used to calculate the e_{sample}. A boxplot and scatterplot where the errors of normal and attack messages were plotted separately together with an ROC plot showing the performance of the NN-AE model as a discriminator are shown in Figure 4.1 and 4.2.

Figure 4.1 shows that the NN-AE is worse performing on the New data set than a classifier that would randomly classify windows as normal or attack with probability 0.5. In fact, the AUC score would increase to $1 - 0.37330 = 0.62670$ only by reversing the predictions of the NN-AE from attack to normal and from normal to attack for each window. Figure 4.2 shows that the NN-AE is almost perfectly able to discriminate between attack and normal windows in the Gear data set, except for a few normal message errors still being larger than attack ones.

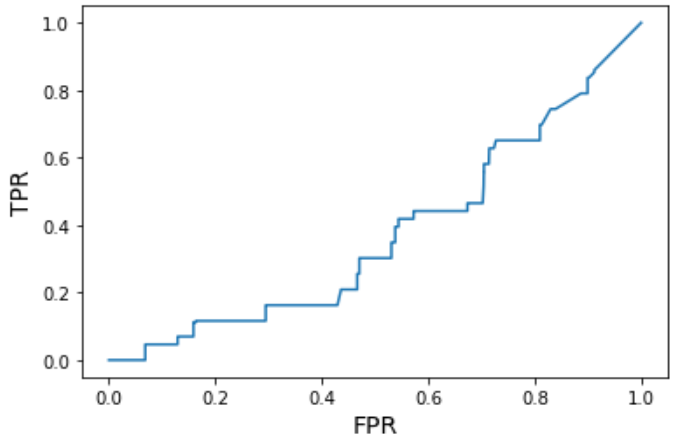
The largest errors of the normal messages were $e_N = 0.03866$ and $e_G = 6.03454$ for the New and Gear data sets respectively. Thus, the thresholds are chosen to be $th_N = 0.05$ and $th_G = 6.2$ for the New and Gear data sets.



(a) Boxplot of the error per message.

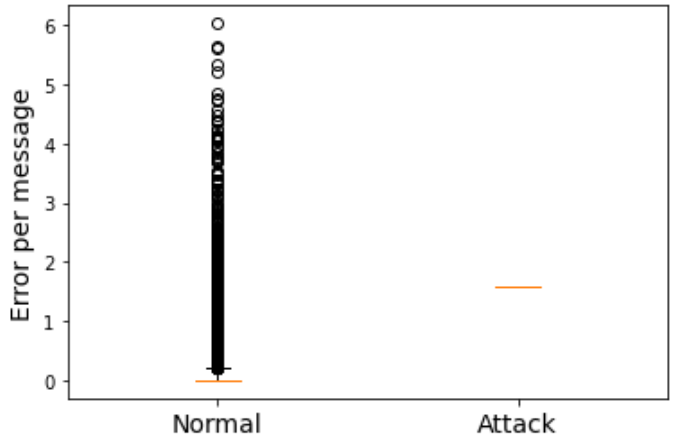


(b) Scatter plot of the errors per message.

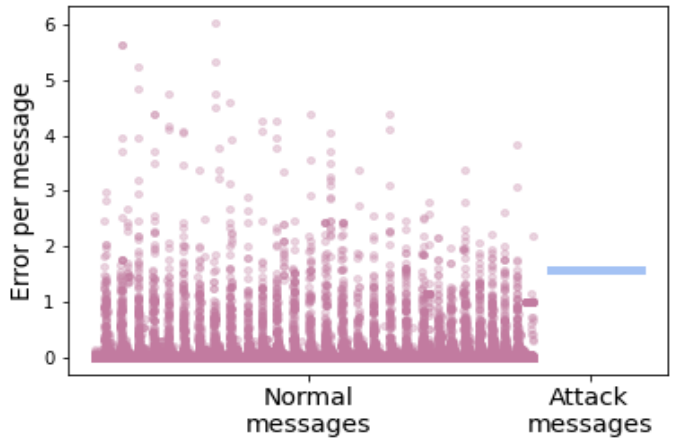


(c) ROC curve obtained from different thresholds between the normal and attack messages. The AUC score is 0.37330.

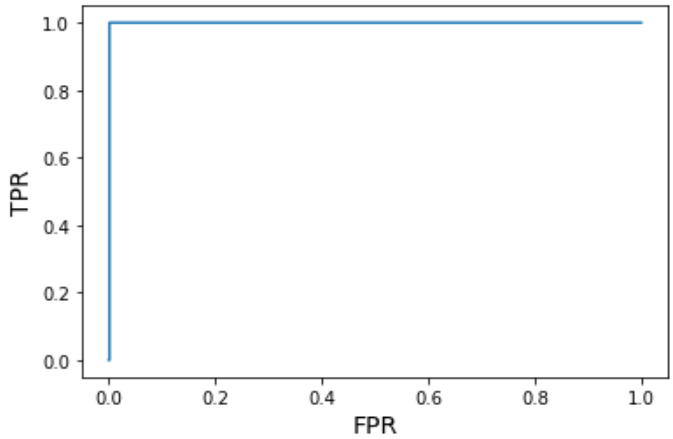
Figure 4.1: Results using NN-AE-3 on the mixed validation subset of the New data set.



(a) Boxplot of the error per message.



(b) Scatter plot of the errors per message.



(c) ROC curve obtained from different thresholds between the normal and attack messages. The AUC score is 0.99901.

Figure 4.2: Results using NN-AE-2 on the mixed validation subset of the Gear data set.

4.1.3 Combination of OCSVM and NN-AE

For clarity, the final selected models and their parameters are shown in Table 4.4. Running this branch on the untouched mixed test subsets gives the metric values in Table 4.5.

Data set	Architecture	Threshold	γ	ν
New	NN-AE-3	0.5	0.0004	8900
Gear	NN-AE-2	6.2	11000	0.066

Table 4.4: The final chosen models and parameters for each data set.

Data set	Accuracy	Balanced Accuracy	TPR	FPR	$F_{\beta=0.1}$	$F_{\beta=10}$
New	1.0	1.0	1.0	0.0	1.0	1.0
Gear	0.88320	0.88417	0.86943	0.10109	0.90718	0.86980

Table 4.5: The final metric scores on the test data sets using Branch 1.

From Table 4.5, it is noted that Branch 1 performs perfectly on the New data set in all metrics. This branch performs well even though the NN-AE part of the branch performed poorly on the mixed validation subset. For the Gear data set, Branch 1 was able to perform well, but far from perfect, even though the NN-AE performed really well on the mixed validation subset. In other words, the NN-AE fails to contribute on the final result of the branch.

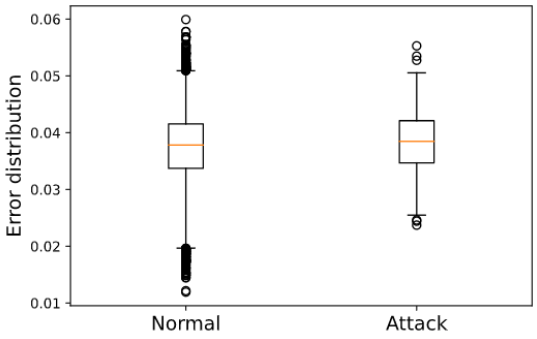
4.2 Branch 2 Results

The results of testing the branch 2 networks on the New and Gear data sets are presented in the form of AUC scores as well as box plots and scatter plots for the reconstruction errors per sample and bit. One sample is one box with 40 messages, or time steps, times 17 times 64, as shown in Figure 3.10, and thus contains $40 \cdot 17 \cdot 64 = 43520$ bits. The normal data is shown on the left in the box plots, and in magenta in the scatter plots. The attack data is shown on the right in the box plots, and in blue in the scatter plots.

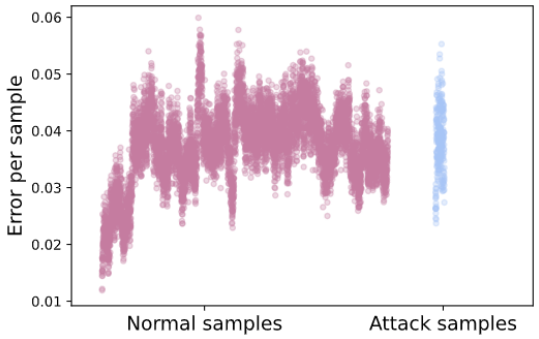
Different networks performed best on the different data sets. The Large LSTM was the best on the New data set and the 2D CNN LSTM was the best on the Gear data set. For comparative reasons, both of these networks were tested on test data from both data sets by optimizing a threshold and calculating metrics. It can be seen in the following Chapters that the networks generally perform much better on the Gear data set than on the New data set.

4.2.1 New data set

The reconstruction errors when testing the networks on the mixed validation subset after training and validating on the normal training and normal validation subsets are shown in figures 4.3 through 4.9 in the form of box plots and scatter plots.

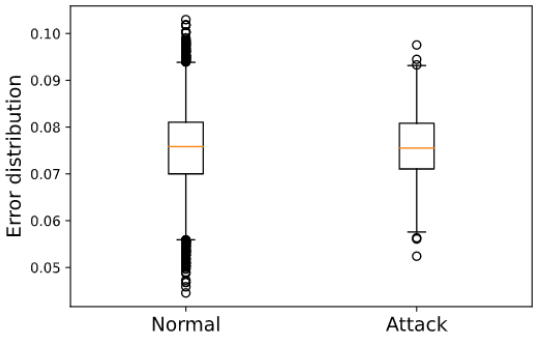


(a) Box plot for Large LSTM reconstruction errors for normal and attack data.

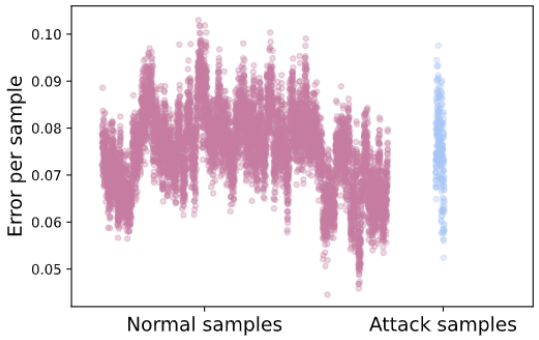


(b) Scatter plot for Large LSTM reconstruction errors for normal and attack data.

Figure 4.3: Plots for Large LSTM reconstruction errors for normal and attack data.

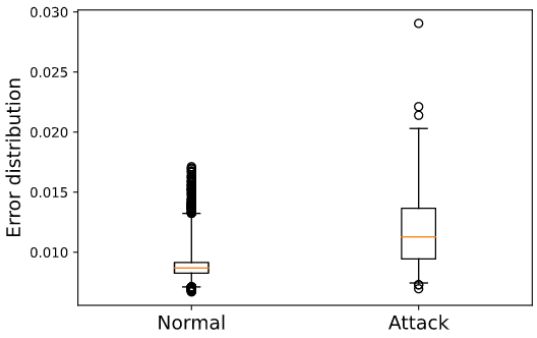


(a) Box plot for Small LSTM reconstruction errors for normal and attack data.

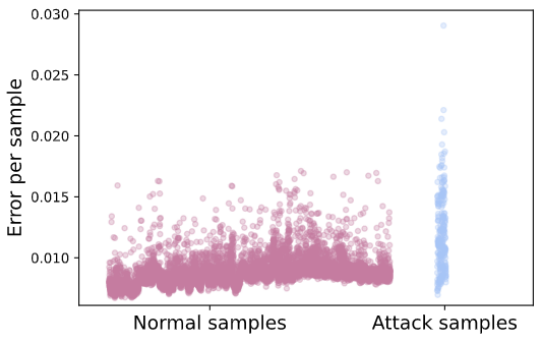


(b) Scatter plot for Small LSTM reconstruction errors for normal and attack data.

Figure 4.4: Plots for Small LSTM reconstruction errors for normal and attack data for the New data set.

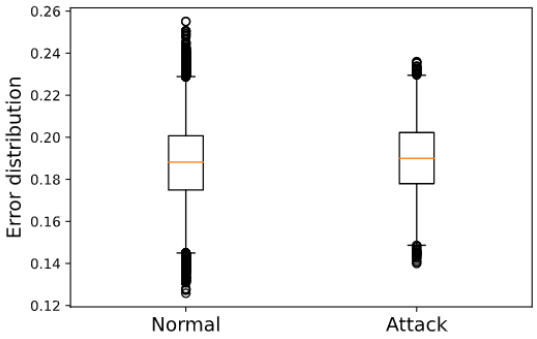


(a) Box plot for 2D CNN LSTM reconstruction errors for normal and attack data.

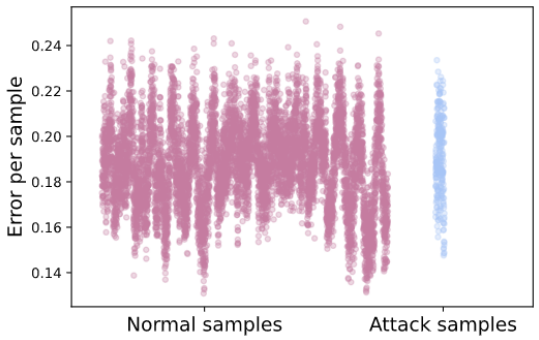


(b) Scatter plot for 2D CNN LSTM reconstruction errors for normal and attack data.

Figure 4.5: Plots for 2D CNN LSTM reconstruction errors for normal and attack data for the New data set.

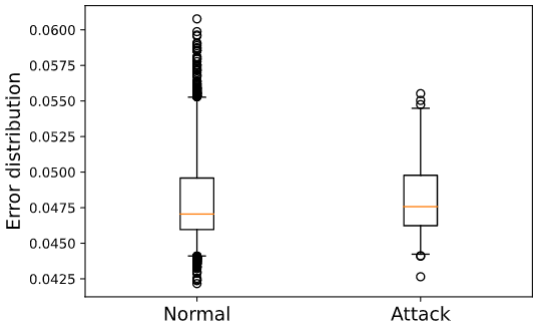


(a) Box plot for 3D CNN reconstruction errors for normal and attack data

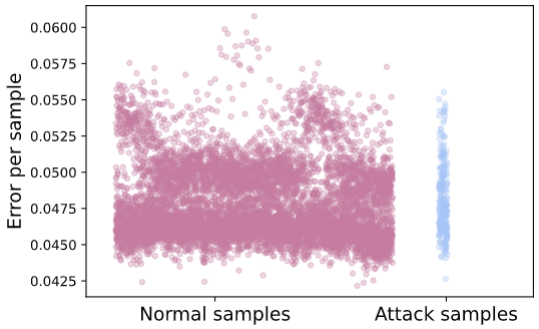


(b) Scatter plot for 3D CNN reconstruction errors for normal and attack data.

Figure 4.6: Plots for 3D CNN reconstruction errors for normal and attack data for the New data set.

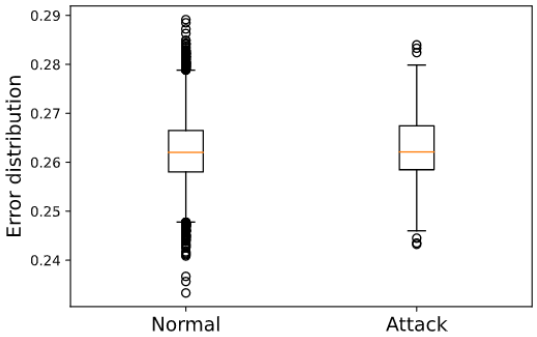


(a) Box plot for TimeDistributed 2D CNN LSTM reconstruction errors for normal and attack data.

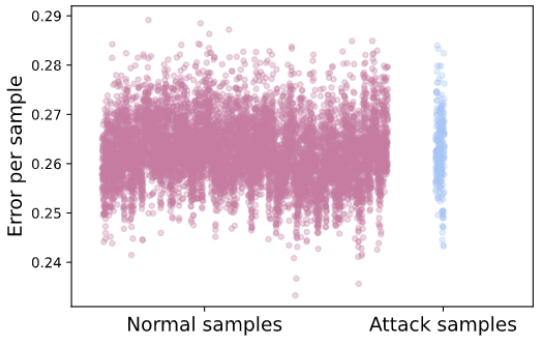


(b) Scatter plot for TimeDistributed 2D CNN LSTM reconstruction errors for normal and attack data.

Figure 4.7: Plots for TimeDistributed 2D CNN LSTM reconstruction errors for normal and attack data for the New data set.

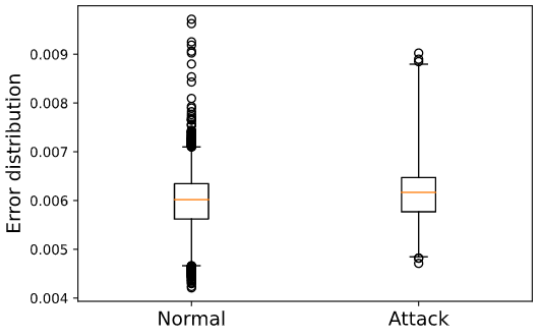


(a) Box plot for Bidirectional ConvLSTM reconstruction errors for normal and attack data.

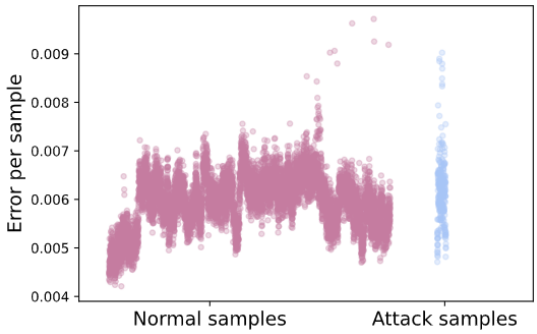


(b) Scatter plot for Bidirectional ConvLSTM reconstruction errors for normal and attack data.

Figure 4.8: Plots for Bidirectional-ConvLSTM reconstruction errors for normal and attack data for the New data set.



(a) Box plot for ConvLSTM reconstruction errors for normal and attack data.



(b) Scatter plot for ConvLSTM reconstruction errors for normal and attack data.

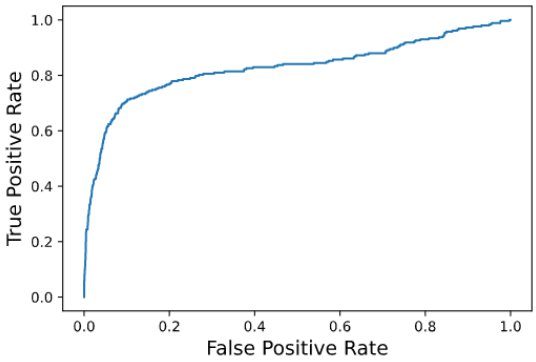
Figure 4.9: Plots for ConvLSTM reconstruction errors for normal and attack data for the New data set.

The AUC scores for the networks are shown in Table 4.6. The best score is shown in bold text.

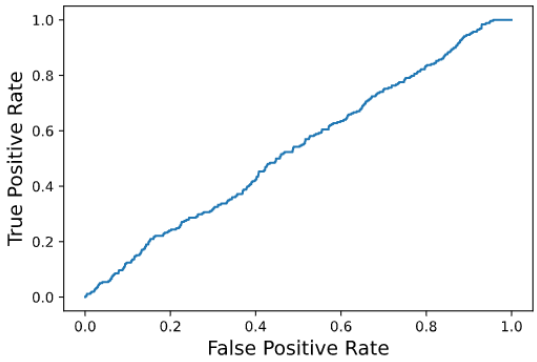
Network	AUC
Large LSTM	0.53247
Small LSTM	0.50707
2D CNN LSTM	**0.82729**
3D CNN	0.54769
TimeDistributed 2D CNN LSTM	0.54476
Bidirectional ConvLSTM	0.51295
ConvLSTM	0.577

Table 4.6: AUC for the different networks. The best result (0.82729 for 2D CNN LSTM) is printed in bold.

As can be seen in Figure 4.5 as well as in Table 4.6, the network that is best at separating normal and attack data for the new data set is 2D CNN LSTM. The ROC curve for this network, , is shown in Figure 4.10 together with the ROC curve for the best performing network on the Gear data set: Large LSTM. The optimal decision thresholds for these networks were decided by taking the threshold in the ROC curve where the True Positive Rate minus the False Positive Rate was as large as possible. The scatter plots for the 2D CNN LSTM and Large LSTM with the chosen thresholds included are shown in Figure 4.11. During inference, all values above the threshold will be labeled as attack and all below will be labeled normal.

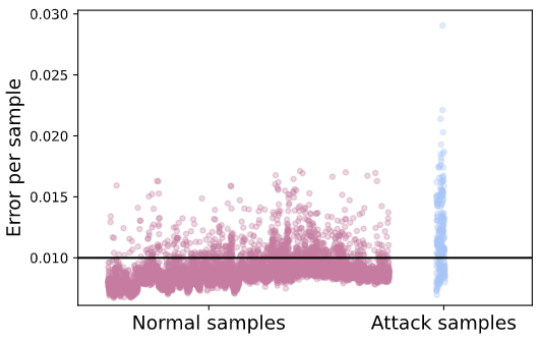


(a) ROC curve for 2D CNN LSTM.

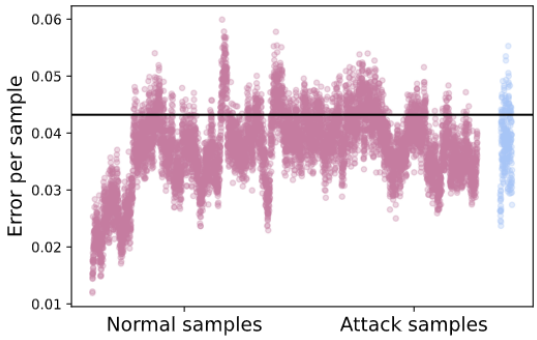


(b) ROC curve for Large LSTM.

Figure 4.10: ROC curves for 2D CNN LSTM and Large LSTM for the New data set .



(a) Error per sample and bit for 2D CNN LSTM with optimal calculated threshold.



(b) Error per sample and bit for Large LSTM with optimal calculated threshold.

Figure 4.11: Error per sample and bit for 2D CNN LSTM and Large LSTM for the New data set with optimal calculated thresholds.

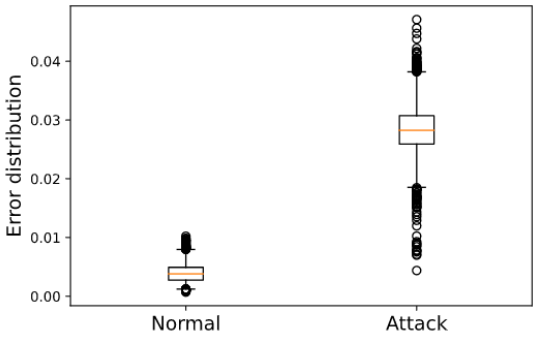
The best performing network 2D CNN LSTM as well as Large LSTM was tested on the mixed test subset of the New data set, using the thresholds optimized with the mixed validation subset to label data samples as normal or attack. The results for this can be found in Table 4.7.

Network	Accuracy	Balanced Accuracy	TPR	FPR	$F_{\beta=0.1}$	$F_{\beta=10}$
2D CNN LSTM	0.89488	0.63407	0.36364	0.09550	0.06505	0.34768
Large LSTM	0.94420	0.51041	0.06061	0.03979	0.02699	0.05986

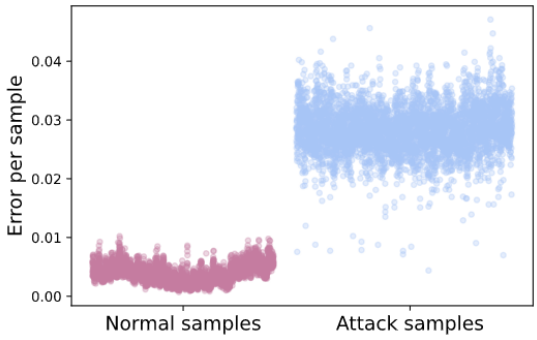
Table 4.7: Evaluation metrics for 2D CNN LSTM and Large LSTM when testing on the mixed test subset using thresholds optimized with the mixed validation subset from the New data set.

4.2.2 Gear data set

The reconstruction errors when testing the networks on the Gear data are shown in figures 4.12 through 4.18 in the form of box plots and scatter plots. The networks were trained and validated on the normal data from the training and validation subsets of the Gear data, and tested on the full validation subset.

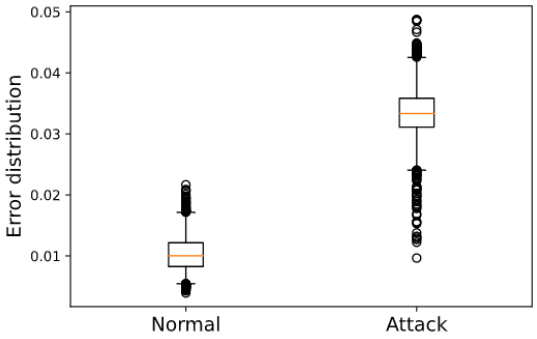


(a) Box plot for Large LSTM reconstruction errors for normal and attack data.

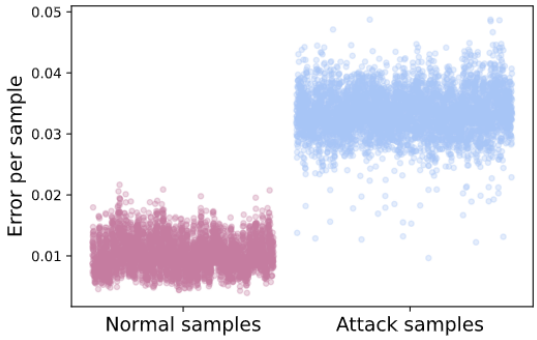


(b) Scatter plot for Large LSTM reconstruction errors for normal and attack data.

Figure 4.12: Plots for Large LSTM reconstruction errors for normal and attack data for the Gear data set.

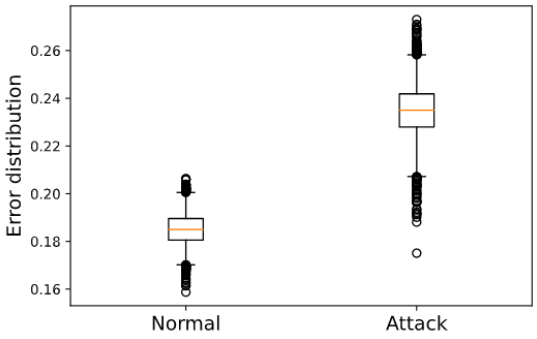


(a) Scatter plot for Small LSTM reconstruction errors for normal and attack data.

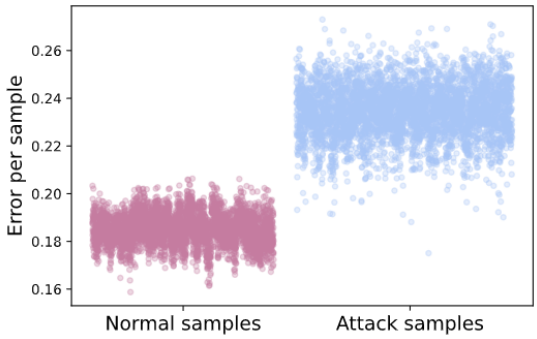


(b) Scatter plot for Small LSTM reconstruction errors for normal and attack data.

Figure 4.13: Plots for Small LSTM reconstruction errors for normal and attack data for the Gear data set.

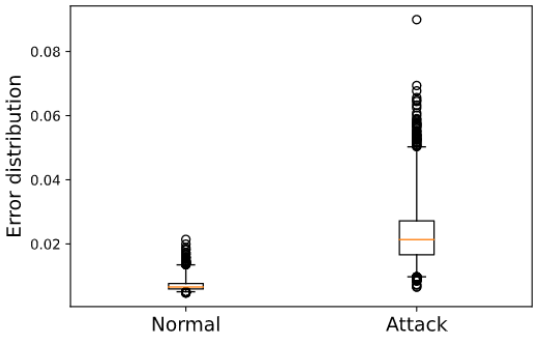


(a) Box plot for Bidirectional ConvLSTM reconstruction errors for normal and attack data.

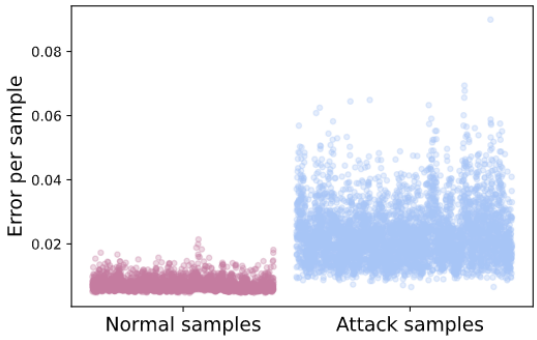


(b) Scatter plot for Bidirectional ConvLSTM reconstruction errors for normal and attack data.

Figure 4.14: Plots for Bidirectional ConvLSTM reconstruction errors for normal and attack data for the Gear data set.

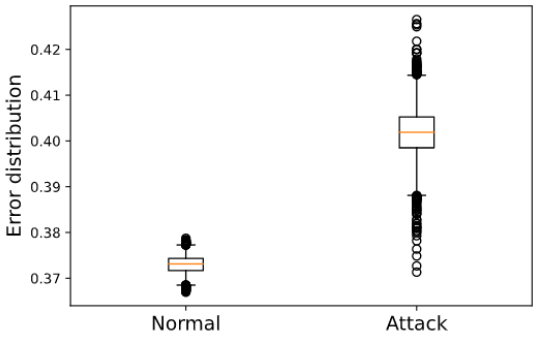


(a) Box plot for 2D CNN LSTM reconstruction errors for normal and attack data.

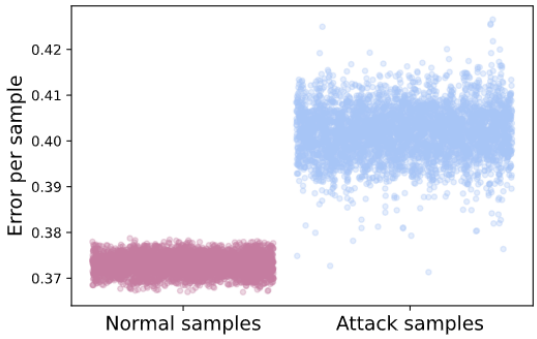


(b) Scatter plot for 2D CNN LSTM reconstruction errors for normal and attack data.

Figure 4.15: Plots for 2D CNN LSTM reconstruction errors for normal and attack data for the Gear data set.

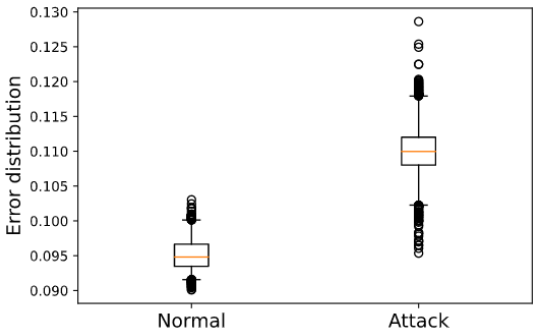


(a) Box plot for 3D CNN reconstruction errors for normal and attack data.

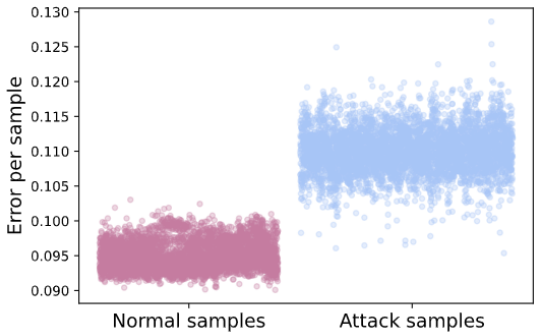


(b) Scatter plot for 3D CNN reconstruction errors for normal and attack data.

Figure 4.16: Plots for 3D CNN reconstruction errors for normal and attack data for the Gear data set.

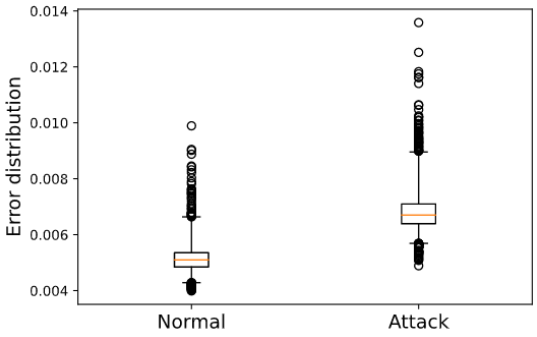


(a) Box plot for TimeDistributed 2D CNN LSTM reconstruction errors for normal and attack data.

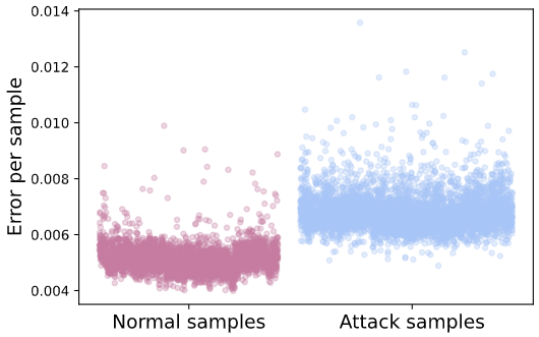


(b) Scatter plot for TimeDistributed 2D CNN LSTM reconstruction errors for normal and attack data.

Figure 4.17: Plots for TimeDistributed 2D CNN LSTM reconstruction errors for normal and attack data for the Gear data set.



(a) Box plot for ConvLSTM reconstruction errors for normal and attack data.



(b) Scatter plot for ConvLSTM reconstruction errors for normal and attack data.

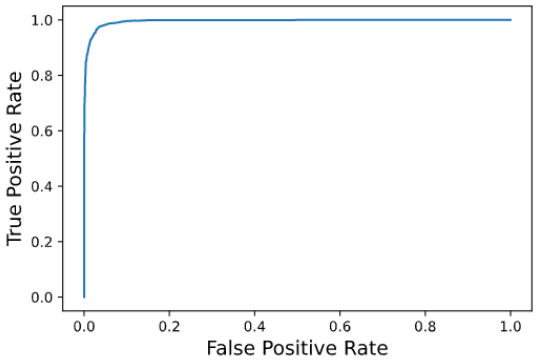
Figure 4.18: Plots for ConvLSTM reconstruction errors for normal and attack data for the Gear data set.

The AUC scores for the networks are shown in Table 4.8. The best score is shown in bold.

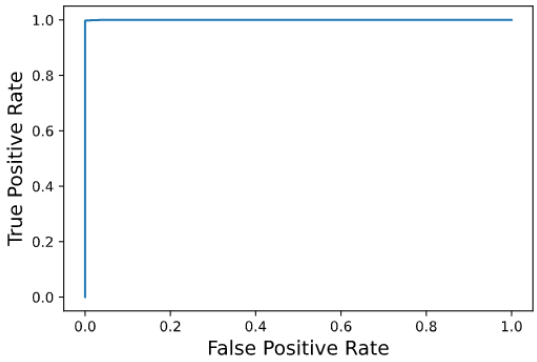
Network	AUC
Large LSTM	**0.99990**
Small LSTM	0.99963
2D CNN LSTM	0.99556
3D CNN	0.99966
TimeDistributed 2D CNN LSTM	0.99949
Bidirectional ConvLSTM	0.99944
ConvLSTM	0.98702

Table 4.8: AUC for the different networks. The best result (0.99990 for Large LSTM) is printed in bold.

As can be seen in Figure 4.12 as well as in Table 4.8, the network that is best at separating normal and attack data for the New data set is Large LSTM. The ROC curve for this network is shown in Figure 4.19 together with the ROC curve for the best performing network on the Gear data set: 2D CNN LSTM. The scatter plots for the 2D CNN LSTM and Large LSTM with the optimal thresholds included are shown in Figure 4.20.

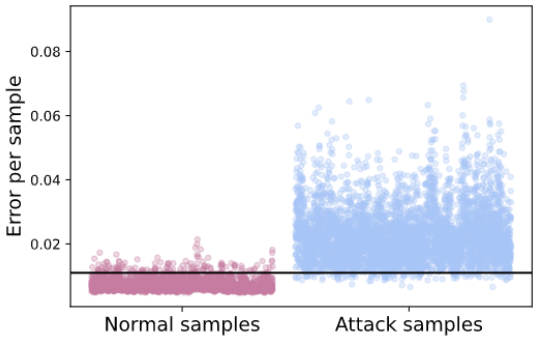


(a) ROC curve for 2D CNN LSTM.

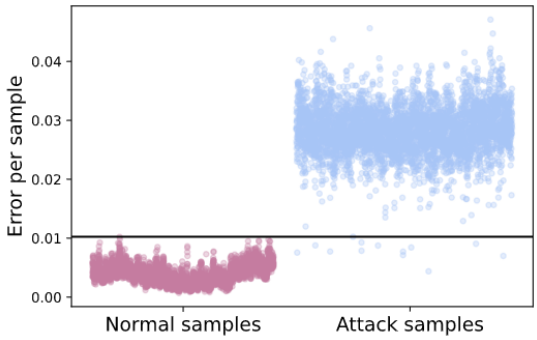


(b) ROC curve for Large LSTM.

Figure 4.19: ROC curves for 2D CNN LSTM and Large LSTM for the Gear data set.



(a) Error per sample and bit for 2D CNN LSTM with optimal calculated threshold.



(b) Error per sample and bit for Large LSTM with optimal calculated threshold.

Figure 4.20: Error per sample and bit for 2D CNN LSTM and Large LSTM for the Gear data set with optimal calculated thresholds.

The Large LSTM and 2D CNN LSTM were tested on the mixed test subset, using the thresholds optimized with the mixed validation subset to label data samples as normal or attack. The results for this can be found in Table 4.9.

Network	Accuracy	Balanced Accuracy	TPR	FPR	$F_{\beta=0.1}$	$F_{\beta=10}$
2D CNN LSTM	0.95173	0.94998	0.97649	0.07653	0.93615	0.97607
Large LSTM	0.99520	0.99507	0.99700	0.00685	0.99404	0.99697

Table 4.9: Evaluation metrics for 2D CNN LSTM and Large LSTM when testing on the mixed test subset using thresholds optimized with the mixed validation subset from the Gear data set.

4.3 Results from combined branches

Combining Branch 1 with Branch 2, where in Branch 2 the 2D-CNN model is used (which performed best on the New data set) using the combinations in Table 3.5 on the mixed test subsets of the New and Gear data sets, the results shown in Table 4.10 were obtained.

	Accuracy	**Balanced Accuracy**	**TPR**	**FPR**	$F_{\beta=0.1}$	$F_{\beta=10}$
New Data set						
Branch1	1.0	1.0	1.0	0.0	1.0	1.0
Branch 2	0.89488	0.63407	0.36364	0.09550	0.06505	0.34768
AND	0.98868	0.68182	0.36364	0.0	0.98297	0.36594
OR	0.90620	0.95225	1.0	0.09550	0.16076	0.95038
Gear Data set						
Branch1	0.88320	0.88417	0.86943	0.10109	0.90718	0.86980
Branch 2	0.95173	0.94998	0.97649	0.07653	0.93615	0.97607
AND	0.91067	0.91476	0.85293	0.02342	0.97512	0.85400
OR	0.92427	0.91940	0.99300	0.1542	0.88126	0.99174

Table 4.10: The metrics scores of the two different data sets using Branch 1, Branch 2 and their combinations OR and AND. The Branch 2 scores are obtained using the 2D CNN LSTM model.

Combining Branch 1 with Branch 2, using the Large LSTM model in Branch 2 (which performed best on the Gear data set) using the same combinations and tested on the same subsets, Table 4.11 was obtained.

	Accuracy	**Balanced Accuracy**	**TPR**	**FPR**	$F_{\beta=0.1}$	$F_{\beta=10}$
New Data set						
Branch1	1.0	1.0	1.0	0.0	1.0	1.0
Branch 2	0.94420	0.51041	0.06061	0.03979	0.02699	0.05986
AND	0.98329	0.53030	0.06061	0.0	0.86695	0.06118
OR	0.96092	0.98010	1.0	0.03979	0.31494	0.97871
Gear Data set						
Branch1	0.88320	0.88417	0.86943	0.10109	0.90718	0.86980
Branch 2	0.99520	0.99507	0.99700	0.00685	0.99404	0.99697
AND	0.92907	0.93336	0.86843	0.00171	0.99680	0.86955
OR	0.94933	0.94589	0.99800	0.10623	0.91547	0.99710

Table 4.11: The metrics scores of the two different data sets using Branch 1, Branch 2 and their combinations OR and AND. The Branch 2 scores are obtained using the Large LSTM model.

5
Discussion and Conclusions

5.1 Branch 1

Branch 1 was able to perform well on both data sets after optimization. It was able to reach balanced accuracy scores of above 0.8 for the Gear dataset and perfect scores for the New data set.

The reason as to why Branch 1 performed perfectly on the New data set was that the 0.04s insertion time of malicious messages was not common in the original Attack-Free data set. The OCSVM, which only uses the time between the messages, was able to leverage this distinction and score perfectly on the data set. The NN-AE was not able to produce large errors for the attack messages. The hypothesis is that the errors are low for the attack data because many of the attack messages are very similar to the normal messages since they were created based on the normal data.

The OCSVM was not able to reach the same performance in the Gear data set. The insertion time of 0.01 of the attack messages in this data set made it hard for the OCSVM to make any clear distinctions. The NN-AE scored very well on the Gear data set, but it was unable to contribute to Branch 1 for two reasons. The first is because this part of the branch is only applied to a window if the OCSVM deems the window as normal, thereby removing its ability to have any decision making on what OCSVM deems as anomaly. The other reason, and the largest one, is due to the threshold being set above the maximum normal message error. Since the maximum message error for the normal messages is $e_G = 6.03454$, it is larger than any attack error. The same argument goes for the New data set. This makes the design of this branch somewhat obsolete. A better approach might have been to have the NN-AE be parallel to the OCSVM and combine the ± 1 decision through an AND or OR operation.

A question arises, is NN-AE necessary to be used at all in the current setting? If we focus on dataset New, comparing the median of the attack and normal errors, it is $3 \cdot 10^{-8}$ and $1.7 \cdot 10^{-7}$ for the attack and normal errors respectively, again aiding the conclusion that using the NN-AE is unnecessary. What would happen if the autoencoder received input very distinct from the any message it had seen before? To test this, 4 data loads shown in Table 5.1 were generated and predicted using the NN-AE-3 model trained previously on the New data set. Their corresponding e_{sample} shown in the same table, show that the errors are larger than normal messages for unknown messages. This motivates the usage of the NN-AE.

Data load label	64 bit pattern	e_{sample}
Simulated Data Load 1	0000111100001111...00001111	6.9
Simulated Data Load 2	0101...0101	4.3
Simulated Data Load 3	10011001...1001	4.5
Simulated Data Load 4	11111111...11111111	9.4

Table 5.1: The packets simulated to use for NN-AE-3 and their errors e_{sample}.

5.2 Branch 2

The results for the branch 2 networks were very different for the different data sets. All networks performed much better for the Gear data set than the New data set, as can be seen by comparing Tables 4.6 and 4.8. One reason for this might be that a distinguishing feature for most of the anomalies in the New data set was the frequency of the messages which none of the branch 2 networks include. It is also interesting to note that the best performing networks were different for the two data sets. In the Gear data set, where only a single ID included anomalous messages, ID was seemingly a superfluous feature, as the best network here was the Large LSTM, which did not include this feature. In the New data set, however, including ID was seemingly necessary, as the two pure LSTM networks did not perform well there. It is difficult, however, to know why the 2D CNN LSTM performed so much better than all the other networks on this data set, since many of them included the same features. The 2D CNN LSTM might have performed better than the TimeDistributed 2D CNN LSTM because the former does not share weights between time steps, while the latter does. Perhaps this allowed the former to capture the time dependent CAN data better. As for the other CNN based networks it is more difficult to speculate. One difference between the 2D CNN LSTM and the others is that it has more trainable parameters, which might have helped it to adapt well to the normal data.

As can be seen in the Figures in Chapter 4.2, some networks that get a lower reconstruction error per sample and bit, still perform worse than others that get a higher reconstruction error per sample and bit when it comes to AUC. This is because they manage to reconstruct not only the normal data well, but also the attack data. For the New data set, the ConvLSTM has a much lower average reconstruction error than all the other networks, but it is only the third best, and not much better than a random classifier, in AUC. Another example is the TimeDistributed 2D CNN LSTM for the two data sets. For the New data set, the normal reconstruction errors for this network (Figure 4.7) are lower than those for the Gear data set (Figure 4.17), but since it also gives low reconstruction errors for the attack data, the AUC for the network is much lower for the New data set (AUC = 0.54476, Table 4.6) than for the Gear data set (AUC = 0.99949, Table 4.6). This is an indication of the complexity of the problem. It is desirable to build a network that can generalize and reconstruct different parts of the CAN data flow well, but not so well as to also reconstruct anomalous data. The nature of the data makes this very difficult, since there can be much variation depending on the activity of the vehicle: is it idling, accelerating, driving carefully on a small winding road or

cruising down the highway? How much variability is normal, and how much could be a sign of a malicious attack?

5.3 The Combined Anomaly Detector

Perhaps the most difficult part of the project is to answer the question: what is an attack? In this thesis, we tried to work around this question as much as possible by focusing mainly on normal data, but some sort of simulated attack data is still needed for validating and testing. As was made clear by the usage of the two different data sets, different types of anomaly detector architectures perform very differently on different types of simulated attacks. This poses the question of whether to try to find a single anomaly detector that performs quite well on both data sets, or whether to always optimize and use the specific architecture that proves most efficient for a specific data set. The danger with the first option is of course that the performance might be lower. This can be seen in Table 4.11 in the chapter for the New data set versus the same chapter in Table 4.10. In Table 4.11 the Large LSTM is used instead of the for the New data set optimal 2D CNN LSTM, whereas in Table 4.10 the 2D CNN LSTM is used. The results in the latter chapter are significantly better. The second option on the other hand, might mean that one would need to spend a large amount of time testing out different configurations before implementing the anomaly detector. Furthermore, different detectors might be optimal for different parts of the data, for example when the car is idling versus when it is driving. Moreover, looking at the Branch 2 results (for example comparing Figures 4.9 and 4.18), it can be seen that there is not a large difference between how well the networks reconstruct the two types of normal data, but the difference rather lies in how well the simulated attacks are reconstructed. This, of course, depends on what type the attacks are, which in a real setting would be unknown. It could thus be dangerous to favour one network over another for a data set based on how well it performs when validating on simulated anomalies. For this reason, branch 2 is made in a way that would make it easy to omit simulated attack data in an actual implementation. Instead of using attack data when optimizing the threshold, one would use only normal data, and set the threshold based solely on this. Some ways to do this could be to set the threshold slightly above the maximum normal reconstruction error, or to assume a Gaussian distribution of the errors and calculate their mean and standard deviation. The threshold could then be set at two or three standard deviations above the mean, and anything above this would be regarded an anomaly. One could also calculate anomaly scores for the errors, as in [51], and decide how large of an anomaly score is allowed before the current error is regarded as indicative of an attack.

It is of course desired for any discriminator to have a perfect false positive and true positive rates of 0 and 1. However, neither Branch 1 nor Branch 2 is able to score perfectly on both data sets. As seen in tables 4.10 and 4.11, high true positive rates go hand in hand with higher false positive rates, since when detectors classify more windows as anomalies there is always a chance that the anomaly actually is a normal window.

The importance of having high TPR is obvious. The higher the rate of TPR, the lower the likelihood of missing an attack, which of course is an important secu-

rity aspect when designing of the different detectors since missing an attack would potentially have serious consequences for the vehicle user.

The importance of FPR is more subtle since having a higher FPR would just mean that more normal messages would be labeled as an attack. Suppose that there is a detector that continuously reads the CAN bus, takes 40 messages at a time and outputs these messages as either being an anomaly or part of the normal CAN flow. The issue with having an FPR rate has to do with the fact that even at a very small FPR, the detector would frequently label the input as an attack. Since the CAN bus outputs approximately 1 million messages every 10 minutes, the number of windows with 40 messages produced is 25000. A false positive rate of for example 0.0001 would a still give at least 2 false positives. If this detector was to be installed in a vehicle and a warning would be sent each time an anomaly was classified, the user would eventually learn to ignore these warnings.

Here is where the Combined Anomaly Detector could be of use. In tables 4.10 and 4.11, it is seen that using the AND operation maintains or even reduces the FPR rate of the best performing branch in both New and Gear data sets, since this operation only outputs an anomaly if both branches agree that it is an anomaly. The same goes for the TPR using the OR operation. In both data sets, the true positive rate is at least maintained compared to the best TPR rate of both branches and for both the New and Gear data sets. Thus, depending on whether an implementer of this detector finds the FPR or TPR to be of more importance, the implementer can control this trade-off by simply choosing between the AND or OR operations.

We conclude that the chosen method is a promising way of detecting anomalies in a CAN bus. Using the combined-branches architecture we are able to provide protection against various types of incoming threats, and for different types of data. The chosen time window of 40 messages corresponds to roughly 1 millisecond which is deemed to be well within the necessary response time. Although further training and fine tuning of the models would be needed before real life implementation, we believe the use of machine learning for CAN bus anomaly detection could lead to safer driving, in an evermore computerized world.

References

[1] *The Rise of Advanced Driver Assistance Systems (ADAS) and Autonomous Driving (AD).* [Online]. Available: https://www.ericsson.com/en/internet-of-things/platform/iot-ecosystem/partners/veoneer

[2] *Automotive, Telecom and ITS companies launch C-V2X trials in Japan.* [Online]. Available: https://www.itsinternational.com/its7/news/automotive-telecom-and-its-companies-launch-c-v2x-trials-japan

[3] *What Is an Electronic Control Unit?* [Online]. Available: https://www.aptiv.com/en/insights/article/what-is-an-electronic-control-unit

[4] M. Di Natale, H. Zeng, P. Giusto, and A. Ghosal, *Understanding and using the controller area network communication protocol: theory and practice.* Springer Science & Business Media, 2012.

[5] M. Levi, Y. Allouche, and A. Kontorovich, "Advanced analytics for connected car cybersecurity," in *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, 2018, pp. 1–7.

[6] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental Security Analysis of a Modern Automobile," in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 447–462.

[7] S. Checkoway, D. Mccoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *USENIX SECURITY*. USENIX, 2011.

[8] C. Valasek and D. Miller, "Remote exploitation of an unaltered passenger vehicle," 2015. [Online]. Available: https://ericberthomier.fr/IMG/pdf/remote_car_hacking.pdf

[9] *Car Hacking Research: Remote Attack Tesla Motors.* [Online]. Available: https://keenlab.tencent.com/en/2016/09/19/Keen-Security-Lab-of-Tencent-Car-Hacking-Research-Remote-Attack-to-Tesla-Cars/

[10] Y. Xun, J. Liu, N. Kato, Y. Fang, and Y. Zhang, "Automobile driver fingerprinting: A new machine learning based authentication scheme," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 2, pp. 1417–1426, 2020.

[11] M. Bozdal, M. Samie, and I. Jennions, "A Survey on CAN Bus Protocol: Attacks, Challenges, and Potential Solutions," in *2018 International Conference on Computing, Electronics Communications Engineering (iCCECE)*, 2018, pp. 201–205.

[12] *As ISIS Promotes Vehicle Attacks, Terrorists Strike In Europe And U.S.* [Online]. Available: https://www.npr.org/sectiowns/thetwo-way/2017/11/01/

561327621/as-isis-promotes-vehicle-attacks-terrorists-strike-in-europe-and-us?t=1612190335893

[13] H. M. Song, H. R. Kim, and H. K. Kim, "Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network," in *2016 International Conference on Information Networking (ICOIN)*, 2016, pp. 63–68.

[14] U. E. Larson, D. K. Nilsson, and E. Jonsson, "An approach to specification-based attack detection for in-vehicle networks," in *2008 IEEE Intelligent Vehicles Symposium*, 2008, pp. 220–225.

[15] M. Müter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, 2011, pp. 1110–1115.

[16] H. Song, J. Woo, and F. F. Li, "In-vehicle network intrusion detection using deep convolutional neural network," 11 2019.

[17] A. Taylor, S. Leblanc, and N. Japkowicz, "Anomaly detection in automobile control network data with long short-term memory networks," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2016, pp. 130–139.

[18] M.-J. Kang and J.-W. Kang, "Intrusion detection system using deep neural network for in-vehicle network security," *PLOS ONE*, vol. 11, no. 6, pp. 1–17, 06 2016. [Online]. Available: https://doi.org/10.1371/journal.pone.0155781

[19] E. Seo, H. M. Song, and H. K. Kim, "GIDS: GAN based Intrusion Detection System for In-Vehicle Network," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, 2018, pp. 1–6.

[20] C. Wang, Z. Zhao, L. Gong, L. Zhu, Z. Liu, and X. Cheng, "A Distributed Anomaly Detection System for In-Vehicle Network Using HTM," *IEEE Access*, vol. 6, pp. 9091–9098, 2018.

[21] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson, "Estimating support of a high-dimensional distribution," *Neural Computation*, vol. 13, pp. 1443–1471, 07 2001.

[22] *Introduction to autoencoders.* [Online]. Available: https://www.jeremyjordan.me/autoencoders/

[23] *ISO 11898-1:2015*. [Online]. Available: https://www.iso.org/standard/63648.html

[24] Larhmam. [Online]. Available: https://commons.wikimedia.org/wiki/File:SVM_margin.png

[25] B. Scholkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, ser. Adaptive Computation and Machine Learning. MIT Press, 2018. [Online]. Available: https://books.google.se/books?id=7r34DwAAQBAJ

[26] Alisneaky. [Online]. Available: https://commons.wikimedia.org/wiki/File:Kernel_Machine.svg

[27] B. Mehlig, "Machine learning with neural networks," 2021.

[28] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural compu-*

tation, vol. 9, pp. 1735–80, 12 1997.

[30] W. Hao, R. Bie, J. Guo, X. Meng, and S. Wang, "Optimized CNN Based Image Recognition Through Target Region Selection," *Optik*, vol. 156, pp. 772–777, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0030402617315735

[31] L. Shang, Q. Yang, J. Wang, S. Li, and W. Lei, "Detection of rail surface defects based on CNN image recognition and classification," in *2018 20th International Conference on Advanced Communication Technology (ICACT)*, 2018, pp. 45–51.

[32] R. Chauhan, K. K. Ghanshala, and R. Joshi, "Convolutional Neural Network (CNN) for Image Detection and Recognition," in *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, 2018, pp. 278–282.

[33] K. Hegde, R. Agrawal, Y. Yao, and C. W. Fletcher, "Morph: Flexible Acceleration for 3D CNN-Based Video Understanding," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 933–946.

[34] A. Diba, A. M. Pazandeh, and L. V. Gool, "Efficient Two-Stream Motion and Appearance 3D CNNs for Video Classification," 2016.

[35] "Temporal 3D ConvNets: New Architecture and Transfer Learning for Video Classification, author=Ali Diba and Mohsen Fayyaz and Vivek Sharma and Amir Hossein Karami and Mohammad Mahdi Arzani and Rahman Yousefzadeh and Luc Van Gool," 2017.

[36] Keras, "Keras documentation: Conv2dtranspose layer." [Online]. Available: https://keras.io/api/layers/convolution_layers/convolution2d_transpose/

[37] M. Zeiler, D. Krishnan, G. Taylor, and R. Fergus, "Deconvolutional networks," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2010*, ser. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2010, pp. 2528–2535, 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2010 ; Conference date: 13-06-2010 Through 18-06-2010.

[38] X. Liu, Q. Zhou, J. Zhao, H. Shen, and X. Xiong, "Fault Diagnosis of Rotating Machinery under Noisy Environment Conditions Based on a 1-D Convolutional Autoencoder and 1-D Convolutional Neural Network," *Sensors*, vol. 19, no. 4, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/4/972

[39] D. Kim, H. Yang, M. Chung, S. Cho, H. Kim, M. Kim, K. Kim, and E. Kim, "Squeezed Convolutional Variational AutoEncoder for unsupervised anomaly detection in edge device industrial Internet of Things," in *2018 International Conference on Information and Computer Technologies (ICICT)*, 2018, pp. 67–71.

[40] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W. kin Wong, and W. chun Woo, "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," 2015.

[41] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *Signal Processing, IEEE Transactions on*, vol. 45, pp. 2673 – 2681, 12 1997.

[42] A. Graves and J. Schmidhuber, "Framewise phoneme classification with

bidirectional LSTM and other neural network architectures," *Neural Networks*, vol. 18, no. 5, pp. 602–610, 2005, iJCNN 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608005001206

[43] S. Siami-Namini, N. Tavakoli, and A. S. Namin, "The Performance of LSTM and BiLSTM in Forecasting Time Series," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 3285–3292.

[44] C. Sammut and G. I. Webb, *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[45] J. P. Mower, "Prep-mt: predictive rna editor for plant mitochondrial genes," *BMC bioinformatics*, vol. 6, no. 1, pp. 1–15, 2005.

[46] *CAR-HACKING DATASET*. [Online]. Available: https://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset

[47] Keras, "Keras." [Online]. Available: https://keras.io/

[48] Tensorflow, "Tensorflow." [Online]. Available: https://www.tensorflow.org/

[49] A. Taylor, N. Japkowicz, and S. Leblanc, "Frequency-based anomaly detection for the automotive can bus," in *2015 World Congress on Industrial Control Systems Security (WCICSS)*, 2015, pp. 45–49.

[50] S. Longari, D. H. N. Valcarcel, M. Zago, M. Carminati, and S. Zanero, "Cannolo: An anomaly detection system based on lstm autoencoders for controller area network," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2020.

[51] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "Lstm-based encoder-decoder for multi-sensor anomaly detection," 2016.

[52] Keras, "Keras documentation: Conv2d layer." [Online]. Available: https://keras.io/api/layers/convolution_layers/convolution2d/

[53] ——, "Keras documentation: Timedistributed layer." [Online]. Available: https://keras.io/api/layers/recurrent_layers/time_distributed/

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY