Rapid, Multi-vehicle and Feed-forward Neural Network based Intrusion Detection System for Controller Area Network Bus

Muhammad Sami
Electrical Engineering
California State University, Long Beach
Long Beach, USA
muhammad.sami@student.csulb.edu

Matthew Ibarra
Electrical Engineering
California State University, Long Beach
Long Beach, USA
matthew.ibarra01@student.csulb.edu

Anamaria C. Esparza
Electrical Engineering
California State University, Long Beach
Long Beach, USA
anamaria.esparza@student.csulb.edu

Saleh Al-Jufout
Electrical Engineering
California State University, Long Beach
Long Beach, USA
saleh.aljufout@csulb.edu

Mehrdad Aliasgari
Computer Engineering and Computer Science
California State University, Long Beach
Long Beach, USA
mehrdad.aliasgari @csulb.edu

Mohammad Mozumdar
Electrical Engineering Department
California State University, Long Beach
Long Beach, USA
mohammad.mozumdar@csulb.edu

***Abstract*—In this paper, an Intrusion Detection System (IDS) in the Controller Area Network (CAN) bus of modern vehicles has been proposed. NESLIDS is an anomaly detection algorithm based on the supervised Deep Neural Network (DNN) architecture that is designed to counter three critical attack categories: Denial-of-service (DoS), fuzzy, and impersonation attacks. Our research scope included modifying DNN parameters, e.g. number of hidden layer neurons, batch size, and activation functions according to how well it maximized detection accuracy and minimized the false positive rate (FPR) for these attacks. Our methodology consisted of collecting CAN Bus data from online and in real-time, injecting attack data after data collection, preprocessing in Python, training the DNN, and testing the model with different datasets. Results show that the proposed IDS effectively detects all attack types for both types of datasets. NESLIDS outperforms existing approaches in terms of accuracy, scalability, and low false alarm rates.**

***Keywords*—**automotive, controller area network, deep neural network, feed-forward neural network, intrusion detection.

I. INTRODUCTION

The Controller Area Network (CAN) Bus is the intercommunication system of vehicles that consists of a series of Electronic Control Units (ECUs) [1]. Each ECU is a subsystem that controls a specific action within the vehicle. The CAN Bus can be compromised through various entry points such as the OBD-II port, Bluetooth, and Wi-Fi. A suitable hacking form may be from local access to the OBD-II port or a remote attack that does not require physical access. ECUs can access data from external networks via intervehicle communication, which in turn creates a communication gateway for possible cyberattacks [2]. Once intruders infiltrate the CAN Bus, they can implement various attacks, such as Denial-of-service (DoS), Impersonation, and fuzzing [3]. Once an injection packet is sent to the system, the vehicle is now more susceptible to future attacks [4], [5]. In literature, a well-liked approach is classifying CAN Bus attacks using Machine Learning (ML) [6] and Deep Learning (DL) [7] algorithms. The approach in [7] uses Deep Belief Network (DBN) in the pre-training phase, extracts the weights and biases from the DBN, and uses a Deep Neural Networks (DNN) to fine-tune the weights and biases. While this method may sound more sophisticated than just using a Feed-Forward Neural Network (FFNN), their approach has two main tradeoffs. The input features used to train the network only consisted of the 8 data bytes, and DBN and DNN require more computing power and, therefore, more training time.

A variety of intrusion1 detection methods have been applied to the CAN Bus of vehicles. Since CAN messages are sent at a fixed frequency, researchers can leverage the timing of CAN packets to detect anomalies [8], [9]. One such method utilizes a Clock-based Intrusion Detection Systems (CIDS) that fingerprints ECUs by estimating the clock skews of CAN messages [3]. However, this method is ineffective against attackers who send messages at the same frequency as the CAN protocol. Other researchers have analyzed specific metrics to determine if changes to packets have occurred. Authors of [10] discussed their algorithm, which measures changes in packet entropy for each CAN Identifier (ID) bit to determine whether an injection attack has occurred and blocks the message. Another intrusion detection algorithm calculates the Hamming distance of attack payloads and compares it to reference Hamming distance data to detect attacks [10]. Other researchers created a mathematical model of CAN Bus data packet to exploit its geometric properties and inform their proof-of-concept intrusion detectors of attacks [11]. A common theme present throughout these methodologies is that they do not require prior insider knowledge of the mapping between CAN IDs and ECU functions. Many previous works have emphasized the importance of creating an algorithm that is both reliable and lightweight [3], [6], [8], [12]. This implies designing an Intrusion Detection System (IDS) that eliminates False Positives (FP) or False Negatives (FN) while at the same time leaving behind a minimal memory footprint to account for ECUs lacking in computational power. This weakness has resulted in limited research being done on

978-1-7281-8744-0/20/$31.00 ©2020 IEEE

ML/DL based algorithms. However, as vehicles have become more connected via Bluetooth, Wi-Fi, and cellular networks, it is assumed that the need for more computationally sophisticated ECUs increases [4]. As a result, researchers have experimented on methods with ML/DL algorithms to secure inner-vehicular networks. One group makes use of a Hidden Markov Model (HMM) to detect CAN Bus attacks [13]. Another team utilized a Hierarchical Temporal Memory (HTM) technique in their anomaly detection system to secure the CAN Bus [12]. More specifically, the HTM was used to predict the flow of CAN Bus data, and its performance was compared to the Recurrent Neural Network (RNN) and HMM models. Another paper mentions an IDS that utilizes a DBN pre-training phase and a DNN to classify in real-time whether a given CAN data is an attack or standard message [7]. This paper discussed creating attack templates of the CAN data to match their DNN to the appropriate set of parameters. However, the downside to this method is that it requires a different model for each CAN ID.

Preceding DNN models were used only to classify attack types. CIDS is used to detect in-vehicle network attacks. In [3], CIDS was useful to decide on fake time stamps and attacks such as fabrication, suspension, and masquerade. Yet, this was only able to be implemented in the design stage, where it cannot predict any future anomalies within the dataset. Another recent approach uses unsupervised DBN with pre-training methods. The top-down process uses the data to create the weight vector and determines the parameter and classification [7]. The results are then used to pre-train the parameters. However, it is inefficient because the parameters have to be continuously fine-tuned. The modified approach used a FFNN, one of the DNN based structures, that can only initialize the weights and is refined later [7].

In this paper, we propose the Network Embedded System Laboratory's Intrusion Detection System (NESLIDS), using the supervised DL algorithm of DNN. The NESLIDS is trained and tested on multiple datasets, including real-time data. Due to the university's constraints, the real-time implementation of NESLID was not performed in a moving or standstill vehicle. This data collection is from diverse vehicles that perform mundane actions in the Los Angeles, CA region. Based on CAN ID and its data, NESLIDS can successfully detect DoS, Fuzzy, and Impersonation attacks with high accuracy. This model can be trained and tested on any vehicle without making any changes in its architecture.

II. The Proposed Intrusion Detection Model

The primary concern with attempting to create an IDS for the CAN Bus of modern vehicles is the semantics of each message. Indeed, in other works [4], [11], much time and effort are devoted to the reverse engineering of CAN IDs and their corresponding data frames. When done thoroughly, it can provide a tentative mapping between CAN data and the function of each ECU. However, this mapping is inconsistent for each make, model, and manufacturer. The process then becomes cumbersome for researchers seeking to derive these relationships manually. To address this concern, in this paper, an intrusion detection method that circumvents this issue entirely is proposed. Fig. 1 gives a pictorial view of the proposed NESLIDS. This method consists of a supervised ML algorithm used to classify whether a given CAN packet is ordinary. Explicitly, the algorithm utilized a FFNN with two hidden layers. The input features are provided by the CAN identifier and the 8 data bytes. Two datasets were acquired to train and test the proposed methodology.

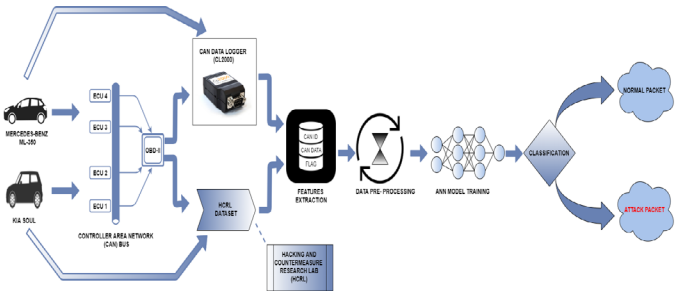

Fig. 1. Methodology adopted in designing NESLIDS

Algorithm 1 defines the data pre-processing procedure needed before implementing DNN training. Both datasets are pre-processed in Python, separating 4 CAN IDs for each. Next, the CAN IDs are extracted out with their respective data bytes and then labeled. The label then pinpoints if the CAN packet is a standard message or an attack. After that, we did the feature extraction normalization using MinMax scaler before shuffling and splitting it up in training and testing to ensure even distribution of standard and attack data. The DNN structure is completed and ready for training.

The weights and biases of NESLIDS are initiated randomly to create a modernized approach. The training phase is split into two sections: forward-feed and reverse-feed. In forward feed mode, the hidden layers are activated sequentially with the Rectified Linear Unit (RELU). In Reverse-Feed mode, the weights and biases are updated through backpropagation using the binary cross-entropy loss function. We favor the Sigmoid as the activation function in the output layer, and Adam as the optimizer for DNN model. Once training is done, test data is then used to predict the output and create a Confusion Matrix (CM) to visualize results.

Algorithm 1: Data Pre-Processing

1. Extract .txt file from CAN data logger
2. Conversion of .txt file into .csv format
3. Import libraries (Sklearn, numpy, pandas)
4. Import Dataset
   i. HCRL **or**
   ii. ML350
5. Extract CAN ID and 8 Data bytes $S_1 = a_1, a_2, \dots, a_9$
6. Generation of DoS and fuzzy attack packet $S_2 = b_1, b_2, \dots, b_9$
7. Create a FLAG for each input and attack vector
8. Concatenate S_1 and S_2 to form dataset DS
9. Extract input matrix X and output matrix Y from DS
10. Convert each $X_i's$ from hexadecimal to decimal where $i = 1,2, \dots, 9$
11. Apply normalization on converted X using MinMax Scaler
12. Encode each Y in 0 or 1 using label encoder
13. Split X and Y into the train and test sets

III. DNN Architecture For NESLIDS

The proposed IDS is a fully connected FFNN with two hidden layers and one output layer. A deep feed-forward network, also recognized as FFNN, is the classic deep learning model. In FFNN, the output of a given layer is not dependent on itself but on the layer before it, i.e., there are no loops or

feedbacks. The FFNN approximates a function f on a given input x by learning parameter θ to compute the closest possible mapping of its output [14]. For each neuron in FFNN, two functions are performed, i.e., pre-activation and activation:

$$pa_n^l = w_{j\,k}^l * x_i + b_n \quad (1)$$

$$a_n^l = g(pa_n^l) \quad (2)$$

where pa_n^l represents the pre-activation of n_{th} neuron in the l_{th} layer; $x_i's$ represent the input vector for the first hidden layer. For the remaining segments, it will take a_n^l from previous layers as input and calculate the results, where $w_{j\,k}^l$ is the weight vector of the k_{th} neuron of the l_{th} layer and the j_{th} neuron of the previous layer; b_n represents the binary value of each neuron. While $g(pa_n^l)$ calculates the activation value using ReLU in hidden layers and Sigmoid in the output layer. Since FFNN is a supervised learning approach, the end goal is to minimize the loss calculated by the activation function in the output layer and compared to the original result.

Our network has an input layer, two hidden layers, and an output layer. The input layer has nine neurons, which correlates to the input vector. The input vector consists of 9 features composed of the CAN ID and 8 bytes of data. The hidden layers have 15 neurons each and become activated by the ReLU activation function. A common rule of thumb in some sources, such as [15], suggests that the number of neurons within the hidden layer should be between the size of the input layer and the output layer. However, authors in [16] strongly disagree and suggest that it would be much better to base the number of neurons on other factors, such as the number of training cases, the amount of noise, and the complexity of the function. In practice, it is found that the number of neurons cannot merely

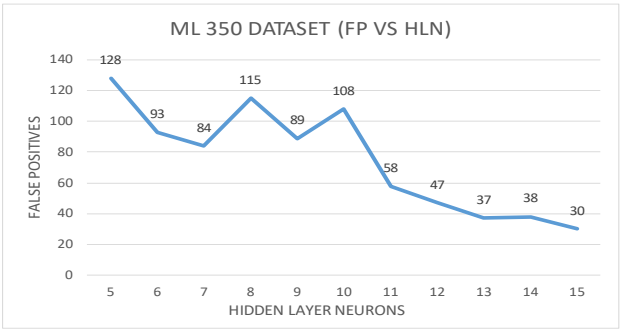

Fig. 2. Hidden Layer Neurons (HLN) impact on False Positives (FP) (ML350 dataset)

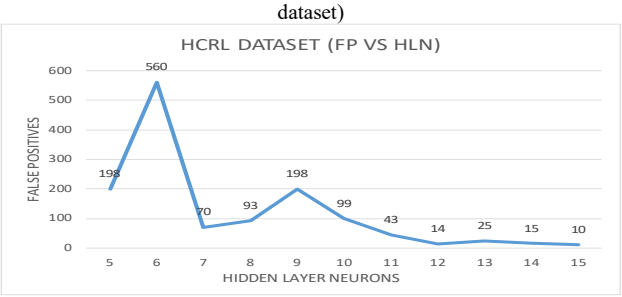

Fig. 3. Hidden Layer Neurons (HLN) impact on False Positives (FP) (HCRL dataset)

be calculated or determined using rules of thumb. It must be tested through trial and error, then compared to other methods. Due to this, the quality of different numbers of hidden layer neurons has been measured out and found that 15 is the best result in terms FP, as can be seen in Fig. 2 and Fig. 3. In other words, the number of FP is minimized at 15 hidden neurons for both HCRL and ML350 datasets. The results for the hidden layer neurons (HLN) greater than 15 are omitted because they were not contributing in reducing FP significantly, instead making NESLIDS computationally expensive. Similar experiment is performed to determine the optimal value of hidden layers.

Furthermore, a dropout value of 0.1 as a method of regularization is used. The dropout rate of 0.1 means that 10% of neurons in that layer is established to zero in that update to prevent overfitting. The output layer consists of only one neuron with Sigmoid as the activation function to yield either 0 for a standard message or 1 for an attack. Fig. 4 represents the proposed network architecture. Each neuron has a bias value, which helps model in fitting for a given dataset.

The proposed network is fully connected, where each neuron is interconnected with other neurons via a weight value. The weights value adopts how quickly the activation function triggers.

Algorithm 2 explains the DNN workflow designed to produce classification results between standard and attack messages. The workflow was used to train models for both datasets without making any changes in the architecture or parameters.

ReLU is an activation function introduced in [17], which has a strong biological and mathematical underpinning. Singular mathematics operations make ReLU computationally less expensive. Moreover, its linearity helps in converging at a faster rate. Another significant advantage of using ReLU is that it doesn't suffer from vanishing gradient problem.

The sigmoid function only produces positive numbers between 0 and 1. Thus, the sigmoid activation function is most useful for training data that is also between 0 and 1. It is essential to state that the activation function is frequently used due to its property of enabling gradient-based optimization methods. This function is crucial as ReLU is not continuously differentiable. The sigmoid function is defined as follows:

$$f(x) = 1/(1 + e^{-x}) \quad (3)$$

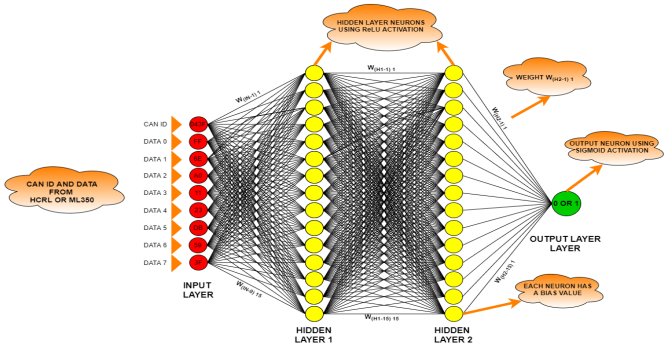

Fig. 4. NESLIDS architecture design implemented separately on the datasets HCRL and ML350. ReLU allows the the network to train effectively.

The sigmoid function parameters influence the speed of backpropagation learning [18]. In this way, it is possible to reduce the attenuation and dilution of the error signal problem, the oscillation problem, and the asymmetrical input problem. In practice, the sigmoid and ReLU activation functions are advantageous over the threshold activation function, since they can be trained using gradient-based methods [19].

Algorithm 2: DNN Workflow

1. Import training sets (X_train, y_train) and testing sets (X_test, y_test)
2. Import library (Keras)
3. Build the Classifier
4. Define the first and second layer
   - Set input and hidden layers neurons
   - Set activation function
   - Set dropout rate of d
5. Define the third layer
   - Set output layer neurons
   - Set activation function
6. Compile the Classifier
   - Set optimizer function
   - Set loss function
   - Set metrics for DNN model
7. Fit Classifier to (X_train, y_train)
   - Set number of epochs
   - Set batch size
8. Input (X_test)
9. Predict output
10. Compare results with (y_test)
11. Create CM to visualize results
12. Observe false positives and false negatives to improve results

Adam optimizer, which is an extension of stochastic gradient descent, is used in the proposed model. It is computationally, efficient and suitable for a large dataset with fewer memory requirements [20]. Individual adaptive learning rates are calculated using the first and second moments of gradients for different parameters [20]. Authors of [21] and [22] suggested Adam as the best option for DL problems. Binary cross-entropy has been used as a loss function, which is also known as Sigmoid Cross entropy loss. One of its essential features is that it calculates the loss for each output vector without getting affected by the result of other vector components [23]:

$$CEL = -\sum_{i=1}^{j=2} r_i \log\big(f(p_i)\big) \quad (4)$$

$$CEL = -r_1\, log\big(f(p_1)\big) - (1 - r_1) log\ (1 - f(p_1)) \quad (5)$$

where CEL represents Cross-Entropy loss; j is the number of classes; r_1 *and* p_1 are real and predicted values of class 1; (1-r_1) and (1-p_1) represent real and predicted values of class 2.

Algorithm 3: NESLIDS Algorithm

Pre-Processing Stage

1. Input: Live data from CAN Bus
   - Apply steps 5, 10 and 11 from Algorithm 1

Detection Stage

2. Load trained DNN model
   - Apply steps 8 and 9 from Algorithm 1

Classification Stage

3. Compare the result to the threshold to classify
   - Notify user in case of an attack message

Algorithm 3 summarizes the NESLIDS algorithm to detect and classify any incoming packet after the DNN model is trained. Since the incoming packet is raw data, some pre-processing techniques are applied before sending it in the detection stage where the CAN packet becomes classified.

IV. DATA

Hacking Countermeasures and Research Lab (HCRL) dataset was obtained from [24]. The other dataset, ML350, was extracted from a Mercedes ML350 using CL2000 CAN data logger via OBD-II port. Another technique of data collection is to connect the CL2000 to a laptop with Wireshark software.

The fuzzy attack dataset from the HCRL website was acquired. Data was then extracted using a KIA Soul vehicle. Then, it was concatenated with self-generated DoS attacks. Messages of 4 unique CAN IDs were extracted and performed hexadecimal to decimal conversion. HCRL dataset has approximately 1.3 million messages.

There are 23 distinct CAN IDs present in the ML350 dataset. Before extracting out 4 CAN IDs from this dataset, the Wireshark plugin CAN Live IDS was used to monitor which CAN IDs are important and more frequently changing during the drive. Approximately one million messages are extracted.

Normalization is performed using the MinMax scaler in a range of (-1, 1). Flag "R" is used for regular messages and "T" for attack messages. Algorithm 4 represents how both datasets are structured. First, sorting is achieved from (M, NF), to extract the desired 4 CAN IDs and create $(R_{sorted}, 9)$. Then the Flag label is added as shown in step 2 to complete the first matrix $(R_{sorted}, 10)$. Following, two separate matrices are then created for DoS and Fuzzy attacks. In the final step, all three matrices were concatenated to produce $(\boldsymbol{M_{concatenated}}, \mathbf{10})$. Each CAN message has nine features. After adding the Flag message, the column dimension becomes 10. To maintain an even distribution of attack and regular messages in the train and test sets, "shuffle=True" is kept while splitting.

Multiple messages can be sent to the CAN network concurrently. The DoS attack sends priority messages to the CAN Bus, commonly with continuous injection. Characteristics of the bus system allow for easy access. The communication between all controllers is accessed from one gateway that manages the protocol of where the message should be sent, received, and translation capability [25].

Algorithm 4: Dataset Structuring

1. Sort $(M, NF) \xrightarrow{yields} (R_{sorted}, 9)$
2. Add Flag label in the sorted matrix $\xrightarrow{yields} (R_{sorted}, 10)$
3. For each CAN ID: Create a DOS attack matrix (D,10)
4. For each CAN ID: Create Fuzzy attack matrix (F,10)
5. Concatenation:

$$\{(R_{sorted}, 10) \mid (D, 10) \mid (F, 10)\} \xrightarrow{yields} (M_{concatenate}, 10)$$

The DoS attack messages target priority CAN message; these are recognized as one with the lowest ID. Priority in a message is based on the seventh bit, where the dominant message contains a '0'. In the dominant state, all other information will be denied access to the bus. DoS attack can continuously change the ID and rapidly increase to its maximum value and preventing any further crucial establishment [1]. Consequently, a malicious attack could target the braking or steering system and could supply a constant DoS. In an Impersonation attack, the CAN system cannot detect the anomaly since the targeted node is unable to differentiate the real and fake packet. Attackers do not have control of the nodes but can inject malicious data to these nodes to induce malfunctions and manipulate the vehicle [26]. In an Impersonation attack, data fields are of regular packets, but the hacker can just replace its CAN ID, which is also going to be one of the legitimate ones. The fuzzy attack sends random identifiers that cause various types of abnormal functions. If implemented with

premeditation, the attack can target the protocol layer of the CAN Bus, initializing a destructive system of fuzzy attacks [7].

V. Results and Discussion

NESLIDS is designed to counter three attacks, namely DoS, Fuzzy, and Impersonation. While training NESLIDS, it was exposed to two attack type packets, excluding Impersonation. For Impersonation attack detection, individual messages were created by using typical data byte fields but replacing it with other CAN IDs included in our dataset.

Results show that NESLIDS was able to identify the Impersonation attack successfully for each CAN ID. Results also shows that each ID while impersonating as another ID, using their data fields, can be detected. This feature also tells that the model is also responsive to single value changes, particularly in CAN IDs.

To evaluate the performance of NESLIDS, we measured the True Positive Rate (TPR) and False Positive Rate (FPR). TPRs and FPRs were evaluated using different threshold values. Then these results were used to produce Receiver Operating Characteristic (ROC) curves followed by calculation of Area Under the Curve (AUC) to determine the performance of ROC. In actuality, AUC summarizes the ROC curve to a single value and helps in interpreting the result unambiguously.

TPRs and FPRs are calculated using True Positive (TP), FP, True Negative (TN), and FN, which can be obtained by creating a CM by comparing the predicted output to actual output. For NESLIDS, when "0", (normal message or positive result), is predicted and detected as "1", (an attack or negative result), then it is classified as an FN. Likewise, when a "1" is detected as a "0", it refers to an FP. TPR is also known as sensitivity of the classification model, while FPR or false alarm rate and is referred to as inverted specificity:

$$TPR\ (\%) = TP/(TP + FN) * 100 \quad (6)$$

$$FPR\ (\%) = FP/(FP + TN) * 100 \quad (7)$$

whereas the accuracy of the model can be calculated by:

$$Acc.(\%) = (TP + TN)/(TP + TN + FP + FN) * 100 \quad (8)$$

These threshold values are carefully selected after validating impersonation attack detection for each CAN ID. If the threshold is set too low, it is highly likely that the impersonation attack will be able to deceive the system. There is a tradeoff between FPR and TPR while selecting a threshold value.

The threshold value needs to be set such that the number of FN is frequently minimized, or else the user will get notifications of fake threats. This notification error can create a severe problem because the user might start ignoring all warnings after receiving spam messages constantly.

Another significant result is AUC from the ROC curve. One of the reasons for selecting the ROC curve is that both datasets are evenly distributed. For imbalance datasets, Precision-Recall curves show better results, and ROC might mislead in that scenario [27]. In the ROC curve, the top left corner represents the ideal condition, which is 100% TPR and 0% FPR. ROC curves can be visualized for both datasets in Fig. 5.

In ML350 plot for a very low FPR TPR is around 60%, but as we start compromising, it immediately reached near 100%.

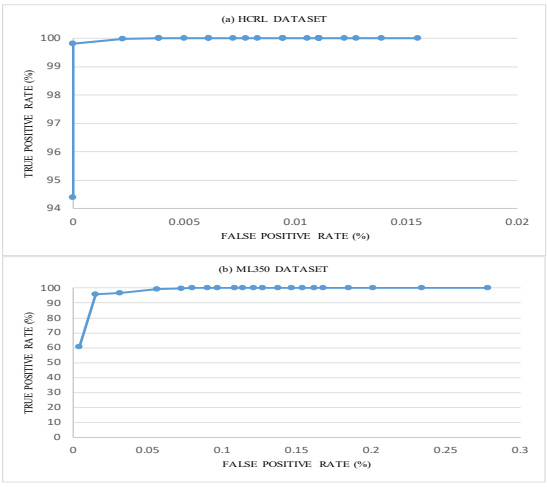

Fig. 5. The Receiver Operating Characteristic (ROC) Curves represented in datasets, (a) HCRL, and (b) ML350.

On the other hand, for the HCRL dataset, TPR is about 94-95% for a very low FPR. AUC for the ML350 is 0.986, while for HCRL, it is 0.99. A higher value of AUC is directly proportional to a better classification of the model. AUC is 0.99 for the HCRL dataset, and this means that the proposed model is working quintessential. The value of AUC is also comparable to the threshold since the threshold can affect the number of FPs and FNs, which corresponds to change in TPRs and FPRs and eventually in ROC curves. The effect of Batch size on accuracy and training time can be visualized in Fig. 6.

As expected, when batch size is increased, training time appears to be faster, but on the other hand, accuracy started to diminish. Since in our system we don't have to train model again, we can compromise on training time and keep accuracy as a priority.

Accuracy for ML350 is 98.6%, while for HCRL is 99.99%. FPs for both models are very low, which is essential in IDS. It is evident that epochs and accuracy are directly proportional to each other, however extensive training can lead to overfitting problems. In this paper, the number of epochs is three. Lastly, Table 1 and Table 2 give a statistical analysis of ML350 and HCRL datasets. These results are calculated by using CM obtained through different thresholds. TP, FP, and FN values from these CMs are used to determine the following results:

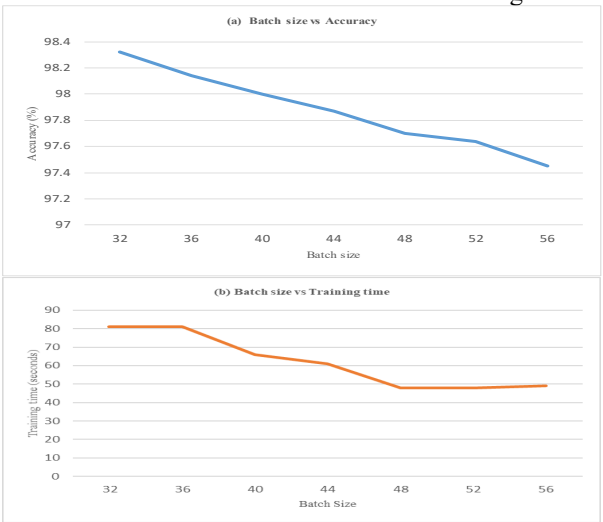

Fig. 6. Performance comparison between various batch sizes. The hidden layer activation is ReLU, while dropout is 0.1, and Epoch is 1.

$$Recall = TP/(TP + FP) \quad (9)$$

$$Precision = TP/(TP + FN) \quad (10)$$

$$F_1\ score = (2*Precision*Recall)/(Precision+Recall) \quad (11)$$

where F_1 score is the harmonic mean of Recall and Precision with the best value of 1 and the worst of 0. Precision is also known as a positive predictive value, whereas Recall tells the sensitivity of the model. F_1 score results from Table 1 and Table 2 are very close to 1, hence indicating the NESLIDS is performing optimally.

TABLE 1
STATISTICS FOR ML350 DATASET FOR DIFFERENT THRESHOLD VALUES

THRESHOLDS	RECALL	PRECISION	F_1 SCORE
0.1	0.993028	0.999329	0.996169
0.15	0.997911	0.999144	0.998527
0.2	0.999244	0.999055	0.99915
0.25	0.999393	0.998936	0.999165
0.3	0.999473	0.998857	0.999165
0.35	0.999493	0.998718	0.999105
0.4	0.999543	0.998659	0.9991
0.45	0.999572	0.998569	0.999071
0.5	0.999592	0.9985	0.999046
0.55	0.999622	0.998381	0.999001
0.6	0.999622	0.998272	0.998947
0.65	0.999642	0.998183	0.998912
0.7	0.999652	0.998093	0.998872
0.75	0.999652	0.998024	0.998837
0.8	0.999652	0.997826	0.998738
0.85	0.999662	0.997628	0.998644
0.9	0.999682	0.997242	0.99846
0.95	0.999751	0.996728	0.998237

TABLE 2
STATISTICS FOR HCRL DATASET FOR DIFFERENT THRESHOLD VALUES

THRESHOLDS	RECALL	PRECISION	F_1 SCORE
0.1	0.999967	0.999967	0.999967203
0.15	0.999972	0.999958	0.999964861
0.2	0.999972	0.999948	0.999960176
0.25	0.999972	0.999948	0.999960176
0.3	0.999972	0.999939	0.999955491
0.35	0.999972	0.999934	0.999953148
0.4	0.999977	0.99993	0.999953148
0.45	0.999977	0.99992	0.999948464
0.5	0.999977	0.99992	0.999948464
0.55	0.999977	0.999911	0.999943779
0.6	0.999977	0.999906	0.999941436
0.65	0.999977	0.999906	0.999941436
0.7	0.999977	0.999906	0.999941436
0.75	0.999977	0.999906	0.999941436
0.8	0.999981	0.999897	0.999939094
0.85	0.999981	0.999892	0.999936752
0.9	0.999981	0.999883	0.999932067
0.95	0.999981	0.999869	0.99992504

VI. CONCLUSION

In this paper, NESLIDS, a simple, rapid and efficient approach to detect malicious behavior in CAN Bus communication was developed. The methodology to implement NESLIDS using CAN data acquired online and through CAN data logger was introduced. Then, a DNN-based FFNN structure, which takes processed CAN data as an input to train its weights and biases and then using test data to produce a binary output, was proposed. Once the NESLIDS is trained on a given dataset and fine-tuned, there is no need to re-train it repeatedly. NESLIDS was tested on two datasets: HCRL and ML350, and practically proved that it could detect DoS, Fuzzy, and Impersonation successfully with high accuracy and low false positives that, to our current knowledge, are not possible in existing techniques using DNN. Future work is needed to implement NESLIDS for real-time automotive intrusions.

REFERENCES

[1] A. Van Herrewege, D. Singelee, and I. Verbauwhede, "CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus," *Proc. 9th Embedded Security in Cars Conf.*, 2011.

[2] I. Studnia et al., "Survey on security threats and protection mechanisms in embedded automotive networks." *Proc. 43rd Annu. IEEE/IFIP Conf. Depend. Syst. Netw. Workshop (DSN W)*, pp. 1-12, 2013.

[3] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," *Proc. 25th USENIX Security Symp.*, pp. 911-927, 2016.

[4] S. Checkoway et al., "Comprehensive experimental analysis of automotive attack surfaces," *Proc. 20th USENIX SEC*, pp. 1-16, 2011.

[5] C. Wang et al., "A distributed anomaly detection system for in-vehicle network using HTM," *IEEE Access*, vol. 6, pp. 9091-9098, 2018.

[6] 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," *Proc. ICOIN*, pp. 63-68, 2016.

[7] 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, 2016.

[8] M. Marchetti and D. Stabili, "Anomaly detection of CAN bus messages through analysis of ID sequences," *Intell. Vehicles Symp. (IV) IEEE*, pp. 1577-1583, 2017.

[9] C. Miller and C. Valasek, "Adventures in automotive networks and control units," *IOActive Labs Research Tech. Rep.*, 2013.

[10] Z. Tyree, R. Bridges, F. Combs, M. Moore, "Exploiting the shape of CAN data for in-vehicle intrusion detection," *Proc. Veh. Technol. Conf.*, 2018.

[11] C. Smith, The Car Hacker's Handbook, 2016, [Online]. Available: http://opengarages.org/handbook/ebook/

[12] Q. Wang, Z. Lu, and G. Qu, "An entropy analysis based intrusion detection system for controller area network in vehicles," *System-on-Chip Conference*, 2018.

[13] S. Narayanan, S. Mittal, and A. Joshi, "Using data analytics to detect anomalous states in vehicles" in *arXiv preprint, 2015.*

[14] I. Goodfellow, Y. Bengio, and A. Courville, in *Deep Learning*, Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org/

[15] A. Blum, "Neural Networks in C++," *An Object-Oriented Framework for Building Connectionist Systems*, 1992.

[16] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemomet. Intell. Lab. Syst.*, vol. 39, no. 1, pp. 43-62, 1997.

[17] J. Freeman and D. Skapura, in *Neural Networks Algorithms, Applications and Programming Techniques*, USA: A. W. Longman, 1991, pp 115-116.

[18] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of back propagation learning," *in From Natural to Artificial Neural Computation*, Berlin Heidelberg: Springer, 1995.

[19] R. Livni, S. Shalev-shwartz, and O. Shamir, "On the computational efficiency of training neural networks," *Advances in Neural Information Processing Systems*, pp. 855-863, 2014.

[20] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[21] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[22] A. Karpathy, "CS321n convolutional neural networks for visual recognition: Course notes," http://cs231n.github.io/

[23] J. Miller, R. Godman, and P. Smyth, "On loss functions which minimize to conditional expected values and posterior probabilities," *IEEE Trans. Inf. Theory*, vol. 39, no. 4, pp. 1404-1408, 1993.

[24] H. Lee, S. Jeong, and H. Kim "OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame," *Privacy Security and Trust*, 2017.

[25] M. Wolf, A. Weimerskirch, and C. Paar, "Security in automotive bus systems," *Proc. Workshop on Embedded Security in Cars*, 2004.

[26] K. Koscher et al., "Experimental security analysis of a modern automobile," *IEEE Symp. Secur. Privacy*, pp. 447-462, 2010.

[27] J. Davis and M. Goadrich, "The relationship between precision-recall and ROC curves," *Proc. 23rd ICML*, pp. 233-240, 2006.