Conversion of an Unsupervised Anomaly Detection System to Spiking Neural Network for Car Hacking Identification

Yassine Jaoudi
Dept. of Electrical and Computer Engineering
University of Dayton
Dayton, OH, USA
jaoudiy1@udayton.edu

Chris Yakopcic
Dept. of Electrical and Computer Engineering
University of Dayton
Dayton, OH, USA
cyakopcic1@udayton.edu

Tarek Taha
Dept. of Electrical and Computer Engineering
University of Dayton
Dayton, OH, USA
tarek.taha@udayton.edu

***Abstract*—Across industry, there is an increasing availability of streaming, time-varying data, where it is important to detect anomalous behavior. These data are found in an enormous number of sensor-based applications, in cybersecurity (where anomalous behavior could indicate an attack), and in finance. Spiking Neural Networks (SNNs) have come under the spotlight for machine learning applications due to the extreme energy efficiency of their implementation on neuromorphic processors like the Intel Loihi research chip. In this paper we explore the applicability of spiking neural networks for in vehicle cyberattack detection. We show exemplary results by converting an autoencoder model to spiking form. We present a learning model comparison that shows the proposed SNN autoencoder outperforms a One Class Support Vector Machine and an Isolation Forest. Furthermore, only a slight reduction in accuracy is observed when compared to a traditional autoencoder.**

Keywords— Autoencoder, Spiking Neural Network, Intrusion detection, Controller area network, Conversion, Loihi, Neuromorphic processor.

I. Introduction

Nowadays huge volumes of data are produced in the form of high-speed streams, which calls for efficient and scalable algorithms for efficient analysis. Real-time analysis of this data to detect anomalies is very useful in many applications including cybersecurity, finance, fault detection, medicine, agriculture, and social media [1]. Anomaly detection aims to discover unexpected events or rare items in data. Time series anomaly detection is a relevant field in computer science and data mining [2-4]. Furthermore, anomaly detection has become essential in the industrial environment, as undetected anomalies can lead to serious damage and revenue loss.

With the development of automotive technology and shift toward autonomous vehicles, advanced electronic devices are installed in vehicles, leading to more complex information traveling throughout the vehicle. Therefore, there is a great amount of research tackling the matter of how to secure the vehicle network, and not putting the driver be at risk due to the malicious attacks performed by hackers. The in-vehicle controller area network (CAN) bus is a standardized serial communication protocol widely used in automobile internal control systems [5]. However, alongside the intriguing benefits of all the recently added functionality in vehicles comes an increased exposure and vulnerability. Attackers could access the automotive network in order to inject messages, manipulate data, or access confidential information. For instance, a hacker could send malicious packets that will result in change of steering wheel position, engage the braking system, or engage acceleration, any of which would deviate from the normal driving behavior of the car's owner. In addition, the Identity Document (ID) in the CAN bus protocol only represents the priority of the message, and there is no original address in the protocol. The receiving electronic control unit (ECU) cannot confirm whether the received data is original data or not. Herein, we seek to develop trainable and deployable systems to detect abuses of communication protocols that do not require retraining the neural network every time a new packet type needs to be classified.

The rest of the paper is structured as follows. We first provide a summary of some previous work in this area in Section II. In Section III, we discuss the fundamentals of the CAN bus communication protocol and the pre-processing of the dataset used in our experiments. In Section IV, we introduce the network architecture used in the anomaly detection system proposed. DNN-to-SNN conversion is performed using the Nengo Loihi framework. Evaluation metrics and results of all performed experiments are also discussed in this section. Concluding remarks are provided in Section V.

II. Related Works

The future of intelligent cars is heading toward the direction of unmanned driving and network communication. An in-vehicle network carries critical messages that are essential vehicle function; thus, the security vulnerabilities in the network are a safety problem. Therefore, the establishment of an in-vehicle information anomaly detection system is extremely necessary.

In the literature [6], a novel intrusion detection system is proposed, which utilizes a DNN for training. This system extracts a feature vector from original CAN data packets, and provides a classification probability that is used to identify normal and abnormal packets. Several studies focused on protecting the in-vehicle network by utilizing the characteristics of the CAN protocol. Each CAN ID has a particular frequency at which the ECU sends messages periodically. According to Miller et al. the frequency of a specific CAN ID does not fluctuate. Thus, they asserted that attacks could easily be detected by monitoring frequency levels [7]. Similarly, Muter et al. [5] proposed an entropy based anomaly detection method. They defined the entropy on the CAN bus and detected attacks by comparing the

entropy to the reference set. Song et al. [8] introduced a light-weight IDS based on timing analysis of the occurrence of CAN messages. Kang et al. [9] used a deep belief network (DBN) structure to construct a classifier and tested it on a simulated dataset. Taylor et al. [10] used a long short-term memory (LSTM) network-based model and tested it on a CAN traffic log collected from a real vehicle. DBN and LSTM networks are generally considered to be more costly while training the model when compared to CNNs. Seo et al. [11] proposed a novel method to train an anomaly detection model using a generative adversarial network. They trained the detection model using normal CAN traffic data and generated noisy data similar to the CAN traffic data. Wang et al. [12] proposed a distributed anomaly detection system using hierarchical temporal memory (HTM). The authors designed a predictor based on the HTM algorithm and used a logarithmic loss function to compute the anomaly score.

III. TIMESERIES DATA

A. Controller Area Network

The controller area network (CAN) is a bus communication protocol which defines a standard for reliable transmission between in-vehicle nodes in real-time developed in 1985 by Bosch. CAN messages such as RPM, steering angle, and current speed, are broadcast through a bus from a transmitter to other nodes with no validation procedure for either the source or the destination. This makes the CAN an easy target for security hackers that can inject messages, leading to vehicle malfunctions. CAN messages are identified by a CAN ID and are composed according to the structure shown in Fig. 1. This structure includes a 1-bit start of frame (SOF) which informs the start of the transmission to all nodes, followed by an arbitration field that uses 11 bits for an identifier (ID) and 1 bit for a Remote Transmission Request (RTR) which is dependent on the kind of CAN frame. Then a 6-bit control field is present that indicates the data length code, followed by up to 8 bytes in a data field that contains the actual transferred data. This is followed by a 16-bit cyclic redundancy code (CRC), which guarantees the validity of the message. A 2-bit acknowledge field (ACK) replaces the ACK part with a dominant bit or retains a recessive bit in the case of a normal message. Lastly, a recessive 7-bit end of frame is present that terminates the message followed by a 3-bit spacing.

1 bit	12 bits	6 bits	0 to 8 bytes	16 bits	2 bits	7 bits	3 bits
Start of frame	Arbitration field	Control field	Data Field	CRC field	ACK field	End of frame	Interframe space

Fig. 1. CAN frame structure.

In this work, we present an anomaly detection system that can detect cyberattacks aimed to cause car malfunction, thus making the CAN bus broadcasting protocol safer. We will be using the Car-Hacking Dataset by HCR Lab [13]. Lee et al. used a real vehicle to build this car hacking dataset. They utilized two customized Raspberry Pi devices for logging network traffic and for injecting malicious packets. They were connected through the OBD-II port to the in-vehicle network and contained four attack types: Denial-of-Service (DoS), a fuzzy attack, drive gear spoofing, and RPM gauge spoofing. For the DoS attack, they injected CAN messages with a CAN ID of 29 zero bits, which is the most dominant CAN ID in the CAN bus, every 0.3 ms. The fuzzy attack is similar to the DoS attack; however, the difference is that the CAN ID and data values of messages are entirely random in the fuzzy attack. In this case random messages were injected every 0.5 ms. The spoofing data were achieved by injecting messages of a certain CAN ID every 1 ms. These messages contained information involving either the drive gear or the RPM gauge depending on the attack applied.

B. Data pre-processing

The packets collected from the Raspberry Pi have the structure shown in Fig. 1, in addition to a timestamp reference in milliseconds. Table. I shows an example of the raw data collected from the Raspberry Pi. The CAN dataset contains three separate text files, one for DoS attacks, one for Fuzzy attacks, and one for Impersonation attacks. In each of these files, normal packets are also present and are randomly interspersed with the attack type.

Table I. Raw CAN data with DoS type of attacks.

Timestamp	ID	DLC	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]
0.0	0000	8	00	00	00	00	00	00	00	00
0.000271	0080	8	00	17	dc	09	16	11	16	bb
0.000495	0000	8	00	00	00	00	00	00	00	00
0.000736	0081	8	40	84	87	00	00	00	00	6b
0.000983	0000	8	00	00	00	00	00	00	00	00

Relative to our proposed system, a number of unessential characters were present in the data that needed to be removed. The most valuable information from the CAN packets are in the arbitration field and the data field, so this was the only data that was used by the system proposed. All other data in the packets was simply removed. Next, the data field can be up to 8 bytes, but when the data field in a given packet contains fewer than 8 bytes, entries were zero padded to uphold data uniformity. This data is hexadecimal, but was converted to decimal processing. When converting the data field, these 8 byte segments were converted to decimal, and each data byte was normalized by column relative to all other packets in the dataset. Then, all data bytes were summed together resulting in a single decimal value.

Overall, pre-processing provides two data in two columns, one for timestamp and one for the data value. This puts the data in a time series format acceptable to the autoencoder model shown in Fig. 2. We built a python function that generates data sequences, combining time steps and contiguous data values to be used as training and testing data. An example input sample that was generated can be seen in Fig. 3.

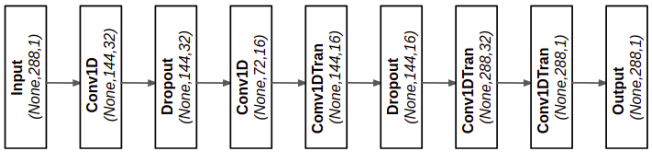


Fig. 2. Autoencoder architecture.

IV. EXPERIMENTAL SETUP

A. Timeseries Autoencoder using Keras

In this section we describe the architecture of the neural network used in this experiment. The architecture implemented to train the network before converting it to an SNN is an autoencoder with one-dimensional convolution layers. The inputs are compressed in the encoder phase of the network produces the encoding at the bottleneck layer. Then the one-dimensional transpose convolution layers in the

decoder section of the autoencoder reconstruct the input only using the encoding produced at the bottleneck layer. The autoencoder is been able to reconstruct the input sample as demonstrated in Fig. 3, and the parameters used in the experiment are provided in Table II.

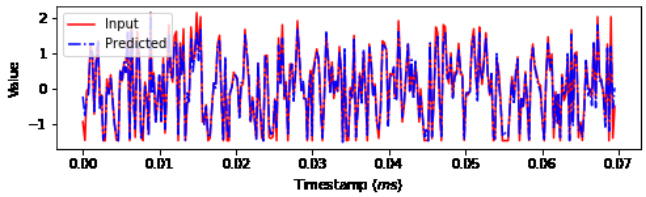


Fig. 3. Input sample and predicted output of AE model.

Table II. Parameter specification employed in simulation.

Parameter	Value
Dropout	0.25
Epochs	50
Batch size	128
Learning rate	0.001
Loss	Mean Squared Error
Optimizer	Stochastic gradient descent

B. *Conversion to Spiking Neural Network*

The DNN-to-SNN conversion approach shows promising results in terms of accuracy and consistency between the original DNN and the converted SNN [14]. As part of this work, we take advantage of NengoDL for converting the Keras based autoencoder model that processes the incoming timeseries data. The NengoLoihi package can also be used for compiling the model to run on Intel's Loihi neuromorphic chip [15]. We converted the Keras-based AE model to spiking form, and executed a simulation using spiking rectified linear neurons, similar to what would be executed on the Loihi chip.

When converting the autoencoder into its equivalent spiking model via NengoDL [16], the toolbox extracts the relevant information from the autoencoder, discarding layers that are irrelevant in the inference phase such as Dropout and Batch Normalization. Then the parsed model is then transformed into a spiking neural network model by applying a normalization process that adapts the weights and biases. Thus, the SNN converted autoencoder is ready to be simulated using NengoLoihi framework.

C. *Pre-Training*

In this system, the training process does not use labels for learning the packets types. The training computation tracks the vector distance D between the input and output samples.

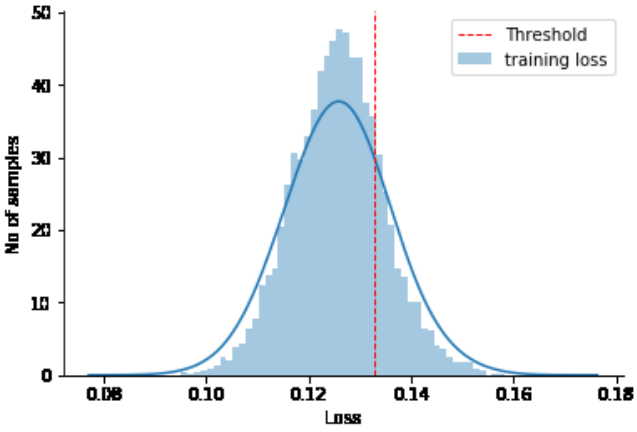


Fig. 4. Mean absolute error training loss and threshold.

Fig .4 provides a mean absolute error loss distribution of the training over all samples and its calculated threshold. If the reconstruction loss for a sample is greater than the computed threshold. Therefore, the model recognizes a pattern that it isn't familiar with which labels it as anomaly.

D. *Evaluation Metrics*

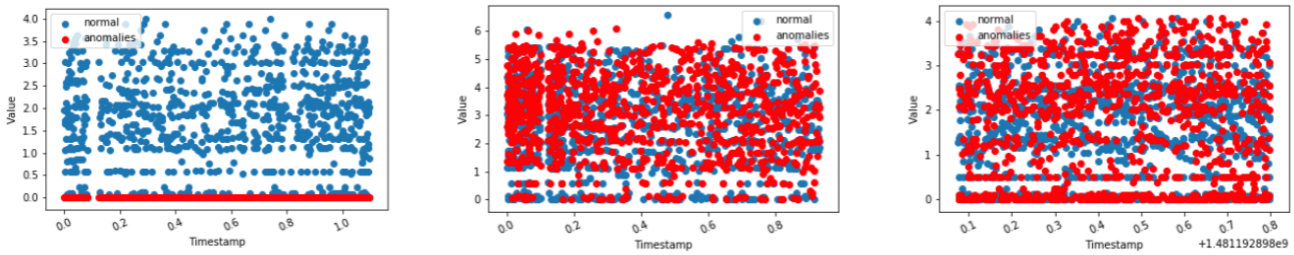
We measured precision, recall, and F1-score for comparison with other algorithms. Precision is the fraction of actual attack frames among the frames detected as attacks. High precision implies a low False Positive (FP) rate. Frequent false alarms annoy and distract users; therefore they should be managed to improve the quality and efficiency of the system. Precision is calculated using the Eq. (1).

$$Precision = TP / (TP + FP) \quad (1)$$

Recall is a fraction of the correctly detected attack frames

Table III. Performance metrics for the neural network algorithms executed for comparison to the proposed system.

	One-class SVM			Isolation forest			Autoencoder			Spiking converted AE		
	Prec.	Rec.	F-sco.	Prec.	Rec.	F-sco.	Prec.	Rec.	F-sco.	Prec.	Rec.	F-sco.
DoS	0.33	0.91	0.49	0.73	0.42	0.53	0.92	0.997	0.96	0.66	0.63	0.64
Fuzzy	0.32	0.91	0.47	0.42	0.63	0.5	0.95	0.72	0.83	0.73	0.93	0.82
Impersonation	0.38	0.99	0.55	0.27	0.55	0.36	0.72	0.40	0.57	0.73	0.92	0.82



(a) DoS attack detection (b) Fuzzy attack detection (c) Impersonation attack detection

Fig. 5. Execution results when executing the proposed SNN autoencoder network.

and represents the true positive rate (TPR). It is computed using Eq. (2).

$$Recall = TP / (TP + FN) \quad (2)$$

TP (true positive) and TN (true negative) are the numbers of packets that are classified correctly as attacks and normal data respectively. FP (false positive) and FN (false negative) are the numbers of packets that are classified incorrectly as attacks and normal data respectively.

The F-score represents a balance between precision and recall. The F-score is typically used to measure classification performance when a dataset has uneven class distribution, and is computed using Eq. (3).

$$F1 = 2 * (Precision * Recall)/(Precision + Recall) \quad (3)$$

V. Results

We evaluated three different models for this application including a One Class Support Vector Machine, an Isolation Forest, and an Autoencoder, in addition to the spiking autoencoder proposed. Once the spiking-model was constructed we evaluated the autoencoder in an inference mode using Nengo Loihi. The performance metrics of the three learning models, as well as the SNN autoencoder are reported in Table III, and the performance of the SNN autoencoder is also visually demonstrated in Fig. 5. Fig. 6 provides an average of the precision, recall, and F-score for each attack type using the four different models.

As Fig. 6 shows the performance, we can clearly see that the autoencoder and its spike-based converted models are being very precise compared to the OCSVM and Isolation forest. When the cost of False Positive is high, this makes our proposed detection system identify a normal packet as an anomaly therefore having a high precision is valuable in our case so that the vehicle will not lose a packet that is necessary.

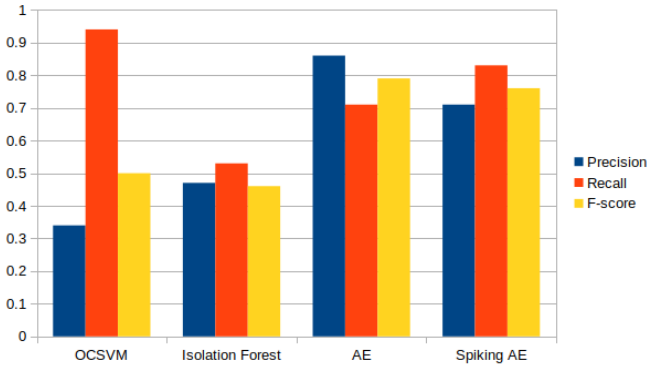


Fig. 6. Performance metrics for the four models compared in this study.

VI. Conclusion

Unsupervised anomaly detection has been studied in a traditional autoencoder, as well as a SNN based autoencoder that was generated using the NengoLoihi framework. The SNN-based model was able to successfully reproduce the performance of the Keras-based autoencoder with a slight reduction in accuracy.

Acknowledgment

The work was supported through the National Science Foundation under grant number 1718633.

References

[1] Ahmad, S., Lavin, A., Purdy, S., & Agha, Z. (2017). Unsupervised real-time anomaly detection for streaming data. Neurocomputing, 262, 134-147.

[2] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM computing surveys (CSUR), 41(3), 1-58.

[3] Fu, T. C. (2011). A review on time series data mining. Engineering Applications of Artificial Intelligence, 24(1), 164-181.

[4] Habeeb, R. A. A., Nasaruddin, F., Gani, A., Hashem, I. A. T., Ahmed, E., & Imran, M. (2019). Real-time big data processing for anomaly detection: A survey. International Journal of Information Management, 45, 289-307.

[5] Müter, M., & Asaj, N. (2011, June). Entropy-based anomaly detection for in-vehicle networks. In 2011 IEEE Intelligent Vehicles Symposium (IV) (pp. 1110-1115). IEEE

[6] Zang, D., Liu, J., & Wang, H. (2018, June). Markov chain-based feature extraction for anomaly detection in time series and its industrial application. In 2018 Chinese Control And Decision Conference (CCDC) (pp. 1059-1063). IEEE.

[7] Miller, C., & Valasek, C. (2013). Adventures in automotive networks and control units. Def Con, 21, 260-264.

[8] Song, H. M., Kim, H. R., & Kim, H. K. (2016, January). 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) (pp. 63-68). IEEE.

[9] Kang, M. J., & Kang, J. W. (2016). Intrusion detection system using deep neural network for in-vehicle network security. PloS one, 11(6), e0155781.

[10] Taylor, A., Leblanc, S., & Japkowicz, N. (2016, October). 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) (pp. 130-139). IEEE.

[11] Seo, E., Song, H. M., & Kim, H. K. (2018, August). Gids: Gan based intrusion detection system for in-vehicle network. In 2018 16th Annual Conference on Privacy, Security and Trust (PST) (pp. 1-6). IEEE.

[12] Wang, C., Zhao, Z., Gong, L., Zhu, L., Liu, Z., & Cheng, X. (2018). A distributed anomaly detection system for in-vehicle network using HTM. IEEE Access, 6, 9091-9098.

[13] Lee, H., Jeong, S. H., & Kim, H. K. (2017, August). OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame. In 2017 15th Annual Conference on Privacy, Security and Trust (PST) (pp. 57-5709). IEEE.

[14] Rueckauer, B., Lungu, I. A., Hu, Y., Pfeiffer, M., & Liu, S. C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. Frontiers in neuroscience, 11, 682.

[15] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. IEEE Micro, 38(1):82–99, 2018.

[16] Rasmussen, D. (2019). NengoDL: Combining deep learning and neuromorphic modelling methods. Neuroinformatics, 17(4), 611-628.