CAN-DO: Exploring Resource-Optimized Temporal Convolutional Networks for Network Anomaly Detection

Sanket Sanjeev Mehrotra
Department of Computer Science
Colorado State University

Kaustubh Jawanjal
Department of Electrical Engineering
Colorado State University

Sudeep Pasricha
Department of Computer Science
Colorado State University

***Abstract*— There are several types of well-performing approaches for intrusion detection in network data in use and in literature today. In this paper, we explore a newer architecture, Temporal Convolutional Networks, and apply an implementation of these networks on a CAN bus dataset in an offline supervised learning approach. We first adapt this dataset for machine learning applications. We then implement a TCN model compare its classification accuracy on this dataset against baseline LSTM architecture. We test the effect of software model optimizations such as model compression, dynamic range quantization, and 16-bit weight quantization on the accuracy of the baseline to check the suitability of this network for IDS applications.**

Keywords*—CAN bus; Anomaly Detection; LSTM; TCN, Supervised Learning, Classification*

I. INTRODUCTION

Innovation in modern cars is centered around efficiently fusing software with hardware. The number of ECU's in modern cars is increased to dozens up to 80 in some cases. These applications are realized as embedded systems and can be characterized as safety-critical (e.g., steering Assist, Adaptive Driver Assistance Systems (ADAS), lane keep assist, collision warning, blind spot warning, parking assist) and not safety critical (Phonebooks and Bluetooth synchronized data are cybersecurity concerns, but not safety-critical).

CAN is the most used protocol in vehicular networks due to its advantages (e.g., reduced wiring cost, error correction, and complexity). It is a broadcast-based communication protocol that supports the maximum baud rate up to 1Mb per second on a single bus. Unauthorized access (attacks) to the CAN bus could incapacitate the vehicle, control it or compromise its safety [6]. CAN is designed for speed and robustness, rather than security and authentication [7]. Therefore, CAN security also requires the development of a feasible, practicable intrusion detection system (IDS) so that appropriate action can be taken as soon as an attack is detected [6]. The primary objective is to detect intrusions into the CAN bus, assuming that the attacker has already gained access to the CAN bus. For example, by gaining unauthorized access to one of the connected ECU's. We further assume that an attacker is now tries to influence the vehicle behavior by manipulating messages [2]. Signature-based IDS can have fast detection times and very few false positives [1]. Signature based methods match traffic patterns against known attacks. They can offer low error rates and can helpfully classify the attacks [6]. Machine Learning-based methods detect anomalies in network traffic, the assumption being that these might indicate an attack. Such methods are more prone to detecting false positives, however, their non-reliance on known attack signatures makes them appealing in novel or unpredictable situations, such as the emerging field of in vehicle cybersecurity [6]. Learning the signature of every known possible attack is not ideal and using signature-based approaches will limit detection to only known attacks. Therefore, it is more advantageous to work with machine learning-based approaches for this type of problem.

A. Problem Description

We aim in this paper to analyze a Anomaly Detection task on controller area network data using LSTMs and TCNs in a supervised manner. We use the OTIDS dataset that is not geared toward ML approaches but adapt and augment the dataset to achieve acceptable accuracy.

B. Contribution

Our approach is the first application that we know of which uses the relatively recently developed Temporal Convolutional Network applied to anomaly detection *specific* to CAN bus data. TCNs have been used for many applications from image classification [12,13] to anomaly detection in time-series data. But most recent papers in the field of in-vehicle anomaly detection are focused on LSTM and Auto-Encoder based approaches [1,2].

II. Related Work

Various techniques are proposed to design IDSs for time-critical automotive systems. The goal of these proposed techniques is to monitor in-vehicle network traffic and detect different types of attacks. We surveyed several papers that seemed similar to our approach or used the same dataset to understand their approaches in building simple and lightweight models for the purpose of detecting anomalies in CAN network data, both in a supervised and unsupervised manner.

C. Metric Based IDS

The authors in [8] proposed an intrusion detection method based on the analysis of the offset ratio and time interval between request and response messages in CAN. The used methodology can detect intrusions by monitoring offset ratio and time interval, and it allows quick intrusion detection with high accuracy. The drawback of this approach is that the methodology proposed in this paper requires additional node arrangement because it sends and receives messages to and from ECUs on the CAN bus [8].

D. Classical ML based IDS

The author of [6] uses one-class compound classifier to detect In-vehicle network attacks. A compound classifier consisted of building a clustering algorithm to determine the boundaries around a class comprising three CAN data fields that had a complex correlation pattern. The drawback of this approach is that class boundaries still left many instances incorrectly classified. The classifier was poor at detecting fuzzy attack, managing to classify only 65%, 52% and 45% of attack records depending on which field was fuzzed.

E. DNN based IDS

The author of [3] proposed a DNN architecture which is an embedded triplet loss network that optimizes the distance between the anchor example and the positive example, makes it smaller than the distance between the anchor example and the negative example, and realizes the similarity calculation of samples, which were originally used in face detection. It is stated in [3] that as compared to traditional anomaly detection methods, the proposed method to learn the parameters with shared-weight could improve detection efficiency and detection accuracy. The drawback of this method is that the performance only improves after increasing the number of hidden layers. However, the increase in the number of hidden layers causes an increase in time consumption of the model. Therefore, it is not ideal because of not having fast inference time. The authors in [4] proposed a literal multi-dimensional anomaly detection approach using the distributed long-short-term-memory (LSTM) framework for in vehicle Control Area Network (CAN). The proposed approach only needs the literal binary CAN message instead of revealing the semantics of CAN message. To enhance the accuracy and efficiency of detection, it detects anomaly from both time and data dimension simultaneously by exploiting multi-task LSTM neural network on mobile edge. Furthermore, the evaluation results show that the proposed mechanism achieves 90% of accuracy, with the assistance of mobile edge, the anomaly detection for one CAN message can be finished within 0.61 milliseconds on average [4]. The results in [4] are promising and it requires less execution time. Hence, we choose to compare our model's performance to a LSTM based model in this paper.

The authors in [1] proposed a lightweight recurrent autoencoder based IDS INDRA using gated recurrent units (GRUs) that monitor messages at a signal level granularity to detect multiple types of attacks more effectively and successfully. The framework is lightweight, has low false positive rate, high accuracy and Fast inference. The results of INDRA were very promising, having good accuracy with low false positives requiring less footprint and requiring less inference time than state-of-the-art prior work. Therefore, Due to its advantages this approach can be used in future experiments. The author in [5] proposed a distributed anomaly detection system using hierarchical temporal memory (HTM) to enhance the security of a vehicular controller area network bus. The HTM model can predict the flow of data in real time, which depends on the state of previous learning. However, the drawback of this approach is that more improvement is needed in the performance of the system in more complex situations and the model needs to be lightweight, requiring less execution time. Furthermore, if the detection system continuously monitors the vehicle network and finds an exception, it appears a new challenge to the real-time system response. Such a response may require an additional design of a separate component [5].

III. Dataset

In this section we describe the dataset we used in our project. We use the OTIDS dataset published by researchers from Korea University in [8], It consists of four datasets, each representing different kinds of attacks on the CAN bus or a steady-state network. For our experiments, we implement Data Preparation, Data Exploration, LSTM and TCN model experiments on these CAN log datasets. Further, we compared the results of these experiments to show which model performs best by using deep learning to learn the normal system behavior in a supervised manner and monitor the CAN bus to check if any anomalies are detected.

F. Types of Attack - Datasets

a) Fuzzy attack : In this attack scenario an attacker can inject messages of randomly spoofed identifiers with arbitrary data, as a result, all nodes will receive lots of functional messages and its causes unintended vehicle behaviors [8]. The attacker can exploit the fuzzy attack by

observing in-vehicle messages and selected target identifiers to produce unexpected behaviors, also when inserting arbitrary data into spoofed identifiers, the authors in [8] found that the steering wheel is shaken vigorously, turn signal lamps light irregularly, instrument pannel blinks in odd manner, and gear shift changes automatically. Unlike DoS attack, it paralyzes functions of a vehicle rather than delaying normal messages via occupancy of the bus.

b) Impersonation attack : As the name suggest impersonating attack is when an attacker can stop message transmission by controlling the target node and can plant/manipulate an impersonating node. If a victim node stops transmitting, all messages sent by the targeted node will be removed from the bus [8]. However, when an ECU receive a remote frame, it is designed to have transmit a data frame immediately, if a receiver node does not receive a response to a remote frame, it can be known that the existing node is attacked or broken [8]. In this case an adversary can plant an impersonating node to respond to a remote frame, Thus, the impersonating node broadcasts data frames periodically and respond to a remote frame like targeted node, to conceal whether each node changes its role [8].

c) DoS attack : DoS attack is when an attacker can inject high priority messages in a short cycle on the bus. Since, any node is not owned or controlled by a network manager, a malicious node may not comply with protocol rules to obtain unauthorized access. DoS attack messages aim at occupying the bus using the theoretically highest priority identifier [8]. Since all nodes share a single bus, increasing occupancy of the bus can produce latencies of other messages and cause threats regarding availability with no response to driver's commands [8].

d) Attack-free : This is an attack free state to help us learn the normal behavior of the CAN frames.

This dataset has both advantages and disadvantages: One advantage is that it is authentic, it is extracted from the real in-car network of a Kia Soul, as compared to other synthetic and generated datasets. One major disadvantage is that it is not labelled in all cases, of the four datasets provided by [8], only two can be confidently labelled to be used in supervised learning. This is because the authors of [8] use an IDS approach targeting time offsets and packet delays in the network and calculating such metrics does not require labels.

This forced us to use only the *Attack-free and DoS datasets* in our approaches.

Table 1: Sizes of the Datasets

Dataset	Size		
	Normal	***Attack***	***Total***
Attack-Free	2,369,398	-	2,369,398
DoS	320,955	335,624	656,579
Fuzzy	448,280	?	591,990
Impersonation attack	515,583	?	995,472
Attack-free + DoS	2,690,353	335,624	3,025,977

```
Timestamp:        0.000000        ID: 0000    000    DLC: 8    00 00 00 00 00 00 00 00
Timestamp:        0.000271        ID: 0080    000    DLC: 8    00 17 dc 09 16 11 16 bb
Timestamp:        0.000495        ID: 0000    000    DLC: 8    00 00 00 00 00 00 00 00
Timestamp:        0.000736        ID: 0081    000    DLC: 8    40 84 87 00 00 00 00 6b
Timestamp:        0.000983        ID: 0000    000    DLC: 8    00 00 00 00 00 00 00 00
Timestamp:        0.001239        ID: 0165    000    DLC: 8    00 08 80 00 00 00 08 80
Timestamp:        0.001484        ID: 0000    000    DLC: 8    00 00 00 00 00 00 00 00
Timestamp:        0.001736        ID: 018f    000    DLC: 8    00 3f 16 00 00 3f 00 00
Timestamp:        0.001984        ID: 0000    000    DLC: 8    00 00 00 00 00 00 00 00
Timestamp:        0.002229        ID: 02a0    000    DLC: 8    62 00 87 9d bc 0c b7 02
Timestamp:        0.002465        ID: 0000    000    DLC: 8    00 00 00 00 00 00 00 00
Timestamp:        0.002654        ID: 02b0    000    DLC: 5    3a ff 00 07 68
Timestamp:        0.002915        ID: 0000    000    DLC: 8    00 00 00 00 00 00 00 00
```

Fig. 1: The original dataset format.

G. Dataset Preparation Pipeline and Insights

Data preparation is done because 1) The datasets that we work with are real CAN logs and the given format in the datasets was not right for direct input to the models, 2) Cleaning the datasets is very important in any machine learning experiments. We spent more time understanding the data versus modelling and predicting because of the complexity of the networks and the problem that we are working with.

Our preprocessing pipeline removed the preset formatting and converted all the hexadecimal fields in the dataset into decimal values. We casted the datatypes of the fields into int32 values and the pandas library automatically converted certain fields to int64 values.

H. Dataset Exploration Insights

Data Exploration is done 1) To check for biases, 2) to understand the data better, 3) to see if we can add labels ourselves for supervised training, 4) to see if we can merge the datasets to build a more suitable, balanced dataset.

We feel that the OTIDS dataset is not tuned for machine learning applications, since all the datasets are on different timestamp axes and the dataset is not labelled at all. We managed to add labels to datasets that we were confident about, but in some datasets, the random nature of the attacks made it impossible to guess what packets were attack packets e.g Impersonation attacks and fuzzy attacks.

I. Unbalanced and Biased Datasets

Exploring the datasets also made us aware of a strenuous fact: the datasets may be a good representative of attacks on a CAN bus, but they were collected separately in different experiments, a usual approach would be to train and test a model on each dataset, but there are some datasets that only contain non-attack data like the Attack-free dataset and some that are split very evenly such as the DoS dataset, both of these datasets introduce bias and uneven class distribution into our models if trained independently.

a) Combining Datasets: One proposed solution to fix this imbalance is to combine the attack-free and DoS datasets. This gives us a resulting dataset where the anomalies are in a more reasonable ratio to the regular network data. This requires that we discard the 'Time-stamp' column in both datasets. Our results show better performance of models on the combined dataset compared to models trained on each dataset alone.

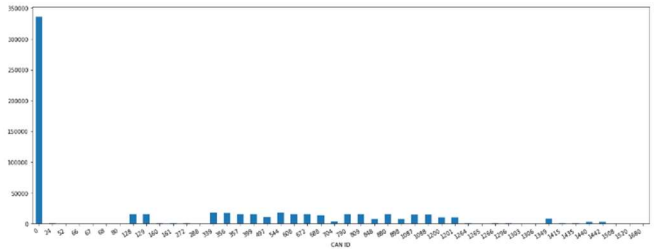


Fig. 2: A CAN ID distribution of the DOS dataset. Notice the huge jump in messages with id 0x00 introducing bias into the dataset

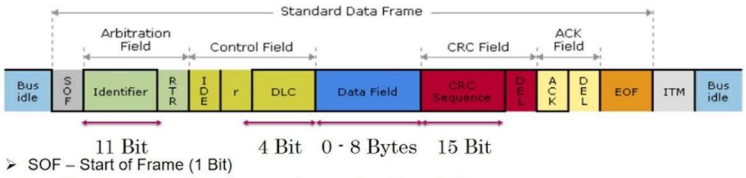
IV. BACKGROUND

J. Anomaly Detection terminology:

An anomaly is "an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism" (Hawkins 1980). Anomaly approaches detect anomalies in the network traffic, the assumption being that these might indicate an attack. Such approaches are more prone to detecting false positives, however, their non-reliance on known attack signatures makes them able to detect both known and unknown attacks but can suffer from higher false positives and relatively slower detection times [1, 6].

K. CAN background :

CAN is a broadcast message-based protocol. In CAN the messages are broadcasted to the existing nodes on the CAN bus and reception filter of each node decides the selection using arbitration identifier of the message. It is the most used communication protocol designed to allow automotive Electronic Control Units (ECUs) to communicate with each other. A CAN message is characterized by a time stamp, an ID and typically 8-byte payload field [8]. CAN protocol was made for fast, fault-tolerant data exchange. CAN has a priority-based arbitration scheme that might allow a malicious node to dominate the bus indefinitely. CAN packets have no authenticator field or source identifier, so a malicious node could indistinguishably send packets to the other nodes [6].



- SOF – Start of Frame (1 Bit)
- Identifier – A message identifier sets the priority of the data frame.
- RTR – Remote Transmission Request, defines the frame type (data frame or remote frame) (1 Bit).
- Control Field – User defined Functions.
- DLC – Data Length Code (4 Bits).
- Data Field – User defined Data (0 to 8 Bytes)
- CRC Field – cyclic redundancy check for Error (Data corruption) detection.
- ACK Field – Receivers Acknowledgement.
- EOF – End of Frame (7 Bits)

Fig. 3: Standard CAN data frame

The built-in security features make CAN robust. It uses a differential wiring to eliminate noise and has two logic levels: logic '0' which is the dominant logic and recessive logic '1' synchronization is provided via signal edges and bit-stuffing. Bit-stuffing rule limits the number of repeated bits [7]. After five consecutive bits of the same logic level, the next bit must be the complement to the previous logic level. If the data has more than five successive corresponding bits, a complement bit is inserted by transmitter CAN controller and it is ignored by the receiver CAN controller, also there are five error checking methods, which are Start of Frame (SOF) single dominant bit for synchronization, Cylindrical Redundancy Check (CRC) checksum of the data for data integrity, Acknowledgement (ACK) bits for successful data transmission, and End of Frame (EOF) bit for stuffing error and frame finalization. CAN protocol is also resilient to physical errors. It can eliminate the faulty nodes from the bus traffic with Error Confinement Mechanism (ECM) [7]. CAN is broadcast Network, so there is no source of destination address and every node can listen to any messages, as the data is not encrypted due to lack of encryption and authentication in CAN bus an attacker may observe, manipulate or understand data leading to privacy issues and injection of faulty data on the system. CAN protocol is vulnerable to DoS (denial of service) attacks, Fuzzy attacks and impersonation attacks, among others. Therefore, these attack datasets were referred to for experiments in this paper.

L. LSTMs :

LSTMs are modified RNNs that use cell state and hidden state information along with multiple gates to remember long term dependencies [1]. LSTM has feedback connections that are not present in standard feedforward neural networks. It not only can process single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks like unsegmented, connected handwriting recognition, speech recognition, and anomaly detection in network traffic or IDSs (intrusion detection systems) [16]. LSTM repeating modules consists of interacting layers that control information flow. LSTM cells are able to selectively track information over many timesteps.

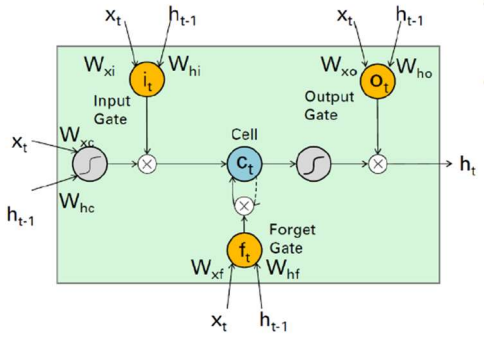


Fig. 4: LSTM cell with different gates

A common LSTM unit consists of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and therefore the three gates regulate the flow of data into and out of the cell. LSTM networks are suitable for classifying, processing, and making predictions supported on time series data, since there are often lags of unknown duration between important events during a time series. LSTMs were developed to affect the vanishing gradient problem which will be encountered when training traditional RNNs. Relative insensitivity to gap length is a plus of LSTM over RNNs, hidden Markov models, and other sequence learning methods in numerous applications [16]. The forget gate is a binary gate that chooses which information to retain from the previous cell state (ct-1). The input gate adds relevant information to the cell state (ct). Lastly, the output layer is controlled by the output gate, which uses information from the previous two gates to produce output [1]. The disadvantage of LSTM is that they are compute intensive due to the addition of multiple gates which causes complicated sequence path. Hence, requiring more memory and runtime.

M. Temporal Convolutional Network :

Temporal convolutional network (TCN) is a framework which employs casual convolutions and dilations so that it is adaptive for sequential data with its temporality and large receptive fields [16]. The TCN architecture can take a sequence of any length and map it to an output sequence of an equivalent length, not unlike an RNN. To take sequence of any length, the TCN uses a 1D fully convolutional network (FCN) architecture, where each hidden layer is that the same length because the input layer, and zero paddings of length (kernel size − 1) is added to keep subsequent layers an equivalent length as previous ones and to achieve no information leakage from the past, the TCN uses causal convolutions, convolutions where output at time t is convolved only with elements from time t and earlier in the previous layer. Advantages of using TCNs for sequence modeling are [17]:

Parallelism*:* Unlike in RNNs where the predictions for later time-steps must await their predecessors to finish, convolutions are often done in parallel since the same filter is used in each layer. Therefore, in both training and evaluation, along input sequence is often processed as a whole in TCN, rather than sequentially as in RNN.

Flexible receptive field size*:* A TCN can change its receptive field size in multiple ways. As an example, stacking more dilated (causal) convolutional layers, using larger dilation factors, or increasing the filter size are all viable options (with different interpretations). TCNs thus affords better control of the model's memory size and are easy to adapt to different domains.

The low memory requirement for training*:* Especially in the case of an extended input sequence, LSTMs and GRUs can easily expend plenty of memory to store the partial results for their multiple cell gates. However, in a TCN the filters are shared across a layer, with the back-propagation path depending only on network depth. Therefore, in practice, gated RNNs are likely to expend to a multiplicative factor more memory than TCNs.

Capturing local information*:* Use of convolution operation helps to capture local information alongside temporal information.

V. METHODS AND EXPERIMENTS

N. Supervised Learning for Anomaly Detection:

The problem with unsupervised learning is that the noise in the data is often been labeled as the as anomalies because the model does not know any better. But if we have domain-specific information about the anomalies then it is better decision to go with supervised learning. As our dataset consists of normal data as well as anomalies our approach is based on supervised learning instead of unsupervised learning.

As stated earlier our approach is the first application that we know of which uses the recently developed Temporal Convolutional Network applied to anomaly detection specific to CAN bus data with the focus being the implementation and exploration of the TCN model on an offline supervised classification application for anamoly detection in vehicular communication (CAN). We also implemented LSTM baseline model for comparison with the TCN performance.

O. LSTM

1) Baseline model

We implement a LSTM baseline model only for the purposes of comparison against the TCN. This lightweight LSTM model is made using the Keras API with 10 cells. It feeds directly into a Dense output layer with two cells to classify the input record into one of the classes: normal or anomaly. We run a few experiments with hyperparameters to try to find the simplest model with the fewest parameters that gives us the best accuracy. This model performs very well, easily ranging in between 98.5-99% training accuracy. Looking at these results, we seem to understand the sheer number of papers trying to leverage this architecture in various IDS approaches encountered by us in literature.

Metric\cells	5	10	20	100
Accuracy(%)	0.9858	0.9918	0.9874	0.9982
Size(MB)	0.015	0.015	0.021	0.17
Total Parameters	152	502	1802	41002
Inference Time(sec)	22	66	25	48

Fig. 5: LSTM Baseline - Number of Cells comparison table.

P. TCN

1) Baseline model

We build a test baseline model to compare against in our exploration of resource constrained TCNs and in our hyperparameter variation experiments. This model is slightly tuned to get a good performance on the test set so that we are sure we are comparing against a model that is not ill-configured. The baseline consists of the following hyper parameters: number of filters = 64, kernel size=2, number of stacks = 1, padding='same' (making this non-causal), use-skip-connections=False (making this a regular convolution) dropout rate = 0.0, activation = 'Relu', kernel initializer = 'he_normal', use batch norm = true.

The model obtains a good performance of ~93% accuracy on our Attack + DoS dataset. This dataset is heavily imbalanced, like most intrusion detection datasets but has a class distribution of ~12.475% anomalies. Seeing this we expect anything better than 88% to be learning to capture some anomalies (roughly 41% of all anomalies).

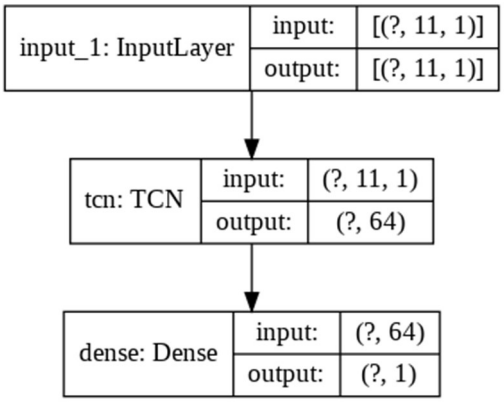


Fig. 6: A plot-model of our baseline TCN

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 11, 1)]           0
_________________________________________________________________
matching_conv1D (Conv1D)     (None, 11, 24)            48
_________________________________________________________________
activation_2 (Activation)    (None, 11, 24)            0
_________________________________________________________________
conv1D_0 (Conv1D)            (None, 11, 24)            72
_________________________________________________________________
batch_normalization (BatchNo (None, 11, 24)            96
_________________________________________________________________
activation (Activation)      (None, 11, 24)            0
_________________________________________________________________
spatial_dropout1d (SpatialDr (None, 11, 24)            0
_________________________________________________________________
conv1D_1 (Conv1D)            (None, 11, 24)            1176
_________________________________________________________________
batch_normalization_1 (Batch (None, 11, 24)            96
_________________________________________________________________
activation_1 (Activation)    (None, 11, 24)            0
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 11, 24)            0
_________________________________________________________________
lambda (Lambda)              (None, 24)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 25
=================================================================
Total params: 1,513
Trainable params: 1,417
Non-trainable params: 96
_________________________________________________________________
```

Fig. 7: An unwrapped TCN layer in a simpler model (1/8th baseline model)

2) Optimized models

In the following sections, we run several experiments on the TCN baseline model to explore the effects of Software Model Optimization on its accuracy, size, and inference time. This is similar to the tasks given to us in lab assignments throughout the course. We highlight that the base model is already quite compact, at 0.4 Mb and inference time ranging from 10-60 secs over several runs. We see a noticeable dip in performance that is expected in quantization.

a) Weight Quantization

We implement dynamic range quantization, in which the weights of the model are statically quantized from floating point to 8-bits of precision. At inference, these weights are converted from 8-bits of precision to floating point and computed using floating-point kernels. This conversion is done once and is cached to reduce latency. We also use the TFLiteConverter class of the TensorFlow Model Optimization tools(tfmot) library to implement other supported quantization, such as the float-16-bit optimization and float-8-bit quantization.

b) Limitations of Weight Pruning and Quantization Aware training

We planned to implement weight pruning and quantization aware training on our baseline model as well but were prevented due to lack of support of tfmot for the TCN layer that was being used. Upon investigation, we found that the TCN layer in the model, though consisting of several supported layers, did not extend the Keras PrunableLayer() API that was used by tfmot to implement certain functions. Similarly, the layers defined in the keras-tcn package could not be wrapped in a quantization aware wrapper that was needed for quantization aware training. A solution to this

problem would require rewriting or extending the keras-tcn module to be usable by tfmot.

Metric\Model	Baseline	Baseline TFLite	float16 optimized	Dynamic Range Optimization
Size(MB)	0.42	0.38	0.21	0.125
Accuracy(%)	0.9391	0.8898	0.8898	0.8898
Total Params	94,273	94,273	94,273	94,273
Inference Time(sec)	10	69	73	139

Fig. 8: The results of implementing Resource Optimized TCNs

3) Ablation Study

In this ablation study, we explored manually tuning certain selected hyperparameters of the baseline to values to compare their effect on the baseline accuracy. TCNs are an incredibly complex network consisting of attention layers, conv-1D layers and a combination of Batch Normalization and Spatial Dropout layers. Our baseline model consists of 6-8 of these blocks, each block consisting of conv1D layers with different levels of dilations.

Thankfully, this implementation of the TCN exposes many hyperparameters of the TCN that we can tweak to experiment with performance. Some of these are seen below. We chose to tune dropout, number of layers/dilations and number of filters. We consciously chose not to vary the kernel size, knowing from previous experience that 2 is an optimal value and inferring that a larger sliding window would only reduce the resolution of the feature maps, possibly making performance worse than it is. Our goal was to find the *simplest* model with the best performance.

a) Effects of Hyperparameters

NUMBER OF FILTERS

This is an important parameter for CNNs (convolutional neural networks), filters are weights that are tuned by running a k size kernel over the input packet data. We use a kernel of size two, meaning a sliding window of size two is being passed over each of our input records to generate a feature map. The number of feature maps generated are treated as input to the next layer and are sometimes pooled together to merge learnt features and reduce the number of weights for the next layer. In the TCN, these feature maps are normalized, and passed through a dropout layer. Our hypothesis is that varying this number will reduce the number of weights and hence size and computation required for training and inference. This will also simplify computation, and which would reduce overfitting if any is occurring in the baseline model.

Metric\Filters	24	64	128
Accuracy(%)	0.9395	0.9391	0.9372
Size(MB)	0.1211	0.4266	1.4732
Total Parameters	14,233	94,273	368,769
Inference Time(sec)	68	10	71

Fig. 9: Ablation over the number of filters in our baseline network

DROPOUT

This is an important hyperparameter that can be used to force the model to drop details of what it learnt and generalize better. Our baseline model worked with 0 dropout, below we introduce drop out layers between Conv1D layers in one 'block' (visible in fig 7). We vary the dropout percentage between 0.5 and 0.8. Both values are ones that have personally given us good values in our previous assignments.

Metric\Droput	0	0.5	0.8
Accuracy(%)	0.9395	0.9357	0.8890
Size(MB)	0.1211	0.1211	0.1211
Total Parameters	14,233	14,233	14,233
Inference Time(sec)	68	66	68

Fig. 10: Dropout hyperparameter variation comparison

LAYERS AND DILATIONS

This hyperparameter controls the depth and skip-connections used in the Conv1D layers in the network. Our baseline uses the default, which is [1,2,4,8,16,32,64], meaning it contains 7 layers of 'blocks' (refer fig 7), each with increasing gaps of 1,2,4 and so on between each kernel convolution step. We noticed that the default layers made our network quite complex, so we decided to drastically simplify the network to see the effect. Looking at our simple input of rows of combinations of 11-12 numeric features, we presumed a simple network to be able to learn just as well as a deep network.

Metric\Layers	[1]	[1,2,4,8]	[1, 2, 4, 8, 16, 32]
Accuracy(%)	0.8890	0.9395	0.9395
Size(MB)	0.0285	0.0845	0.1211
Total Parameters	14,233	14,233	14,233
Inference Time(sec)	41	62	68

Fig. 11: Layer Dilation experiments comparison table

VI. LIMITATIONS AND FUTURE WORK

Our work is focused on implementing and exploring TCNs on an offline supervised classification application. We also implement a baseline LSTM model for comparison with the TCN performance. Both the TCN and LSTMs are famous for

their ability to predict sequences by remembering a number of steps behind. It would be within the scope of the future work of this paper if we choose to explore this problem of IDS in a seq2seq manner. We would also like to explore other CAN bus datasets, as we found our chosen dataset to not be geared towards machine learning applications. We managed to get a relatively balanced dataset by combining two of the offered datasets, namely the attack-free and DoS datasets, giving us a 12-13% anomaly to normal packet ratio.

VII. CONCLUSIONS

Through our experiments exploring resource constrained TCNs for the purpose of intrusion detection, we have focused the scope of our network on detection of DoS attack packets in a CAN network. Our dataset is heavily biased and unevenly distributed and combining two datasets slightly alleviated the bias.

Our TCN model results are encouraging and show potential to improve given more hyperparameter tuning. The best TCN that we configured managed to achieve an accuracy of 93.95%, with the simplest LSTM model easily achieving a 96%. These results promote further exploration to extract more competitive performance from the TCNs.

Our experiments in software model optimizations on TCNs show that there is a drop in performance of around 5% accuracy upon applying quantization and compression methods to our baseline, along with reductions in size of about ~30%. These results are expected but the corresponding increase in inference time to up to 193s show that this model may not be applicable for real-time inference.

We investigated ways to correctly learn from an imbalanced dataset and plan to try our experiments again after oversampling our datasets using the SMOTE algorithm, present in the *imblearn* package. Also, we are aware of more concise metrics to measure the performance of our models such as F1-score, recall and precision and we were only able to try out one of these on our results.

We ran into many compatibility issues between TensorFlow Model Optimization Tools and the LSTMs and TCN models that we had built, but we were able to achieve our goal learning how to work with TCNs. Reading about the TCN dilations, stacking and skip-connections was very informative and going over the literature, we discovered a whole field of important work that is key in defining the future of the automotive industry, both autonomous and manual.

All the code, data and documents/resources can be found at mehrotrasan16/CS581-CAN-DO-Project (github.com)

REFERENCES

[1] [V. K. Kukkala, S. V. Thiruloga and S. Pasricha], INDRA: Intrusion Detection Using Recurrent Autoencoders in Automotive Embedded Systems in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 11, pp. 3698-3710, Nov. 2020, doi: 10.1109/TCAD.2020.3012749.

[2] [Hanselmann, M., Strauss, T., Dormann, K., & Ulmer, H. (2020)]. CANet: An Unsupervised Intrusion Detection System for High Dimensional CAN Bus Data. IEEE Access, 8, 58194-58205.

[3] [Zhou, A.; Li, Z.; Shen, Y.], Anomaly Detection of CAN Bus Messages Using a Deep Neural Network for Autonomous Vehicles. Appl. Sci. 2019, 9, 3174.

[4] [K. Zhu, Z. Chen, Y. Peng and L. Zhang], Mobile Edge Assisted Literal Multi-Dimensional Anomaly Detection of In-Vehicle Network Using LSTM in IEEE Transactions on Vehicular Technology, vol. 68, no. 5, pp. 4275-4284, May 2019, doi: 10.1109/TVT.2019.2907269.

[5] [C. Wang, Z. Zhao, L. Gong, L. Zhu, Z. Liu and X. Cheng], A Distributed Anomaly Detection System for In-Vehicle Network Using HTM in IEEE Access, vol. 6, pp. 9091-9098, 2018, doi: 10.1109/ACCESS.2018.2799210.

[6] [Tomlinson, Andrew & Bryans, Jeremy & Shaikh, Siraj. (2018)], Using a one-class compound classifier to detect in-vehicle network attacks. 1926-1929. 10.1145/3205651.3208223.

[7] [M. Bozdal, M. Samie and I. Jennions], "A Survey on CAN Bus Protocol: Attacks, Challenges, and Potential Solutions," 2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE), Southend, United Kingdom, 2018, pp. 201-205, doi: 10.1109/iCCECOME.2018.8658720.

[8] [H. Lee, S. H. Jeong and H. K. Kim], OTIDS: A Novel Intrusion Detection System for In-vehicle Network by Using Remote Frame 2017 15th Annual Conference on Privacy, Security and Trust (PST), Calgary, AB, 2017, pp. 57-5709, doi: 10.1109/PST.2017.00017.

[9] [J. Schneible and A. Lu], "Anomaly detection on the edge," MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM), Baltimore, MD, 2017, pp. 678-682,
doi: 10.1109/MILCOM.2017.8170817.

[10] https://github.com/locuslab/TCN - (Accessed on 28th Nov 2020).

[11] https://github.com/philipperemy/keras-tcn#keras-tcn – (Accessed on 28th Nov 2020).

[12] https://github.com/ashishpatel26/tcn-keras-Examples/blob/master/TCN_cifar10.ipynb - (Accessed 7th Dec 2020)

[13] https://github.com/ashishpatel26/tcn-keras-Examples/blob/master/TCN_MNIST.ipynb - (Accessed 7th Dec 2020)

[14] https://beckernick.github.io/oversampling-modeling/ - (Accessed 12th Dec 2020)

[15] [Bai, S., Kolter, J. Z., & Koltun, V. (2018)]. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271.

[16] https://en.wikipedia.org/wiki/Long_short-term_memory

[17] Yangdong He and Jiabao Zhao 2019 J. Phys.: Conf. Ser. 1213 042050

[18] CS581-C1 Assignment #3 – Software Model Optimizations Report/Code.

[19] CS581-C1 Assignment #1 – Software Model Optimizations Report/Code.

[20] Our code repository mehrotrasan16/CS581-CAN-DO-Project (github.com)