Intrusion Detection for Time-Series IoT Data with Recurrent Neural Networks and Feature Selection

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

06-01-2021 / 08-01-2021

CITATION

Syed, Naeem Firdous; Ge, Meng Meng; Baig, Zubair (2021): Intrusion Detection for Time-Series IoT Data with Recurrent Neural Networks and Feature Selection. TechRxiv. Preprint.
<https://doi.org/10.36227/techrxiv.13525409.v1>

DOI

[10.36227/techrxiv.13525409.v1](https://doi.org/10.36227/techrxiv.13525409.v1)

Intrusion Detection for Time-Series IoT Data with Recurrent Neural Networks and Feature Selection

Naeem F. Syed, Mengmeng Ge and Zubair Baig

{naeem.syed, mengmeng.ge, zubair.baig}@deakin.edu.au, Deakin University, Geelong, VIC, Australia

Abstract—Deep learning (DL) techniques are being widely researched for their effectiveness in detecting cyber intrusions against the Internet of Things (IoT). However, time sensitive Critical Infrastructures (CIs) that rely on IoT require quicker detection of cyber intrusions close to the constrained devices in order to prevent service delays. FoG layer with higher computing resources and close proximity to IoT devices makes them suitable to deploy deep learning techniques for effective intrusion detection. However, communication overheads due to large volume of IoT data and computation requirements for deep learning models prevents effective use of FoG layer and raises scalability issues. To counter these issues, we propose a novel IoT intrusion detection framework incorporates feature selection step on time-series IoT data, followed by a deep learning Recurrent Neural Network (SimpleRNN and Bi-directional Long Short-Term Memory (LSTM)) based IoT attack detection. The effectiveness of the proposed approach was evaluated using the high-dimensional BoT-IoT dataset which contains large volumes of realistic IoT attack traffic. Results show that feature selection methods significantly (90%) reduced the dataset size and still achieved an increased recall rate compared to full feature set without losing class differentiation ability. The SimpleRNN and Bi-LSTM models also did not suffer any underfitting or overfitting with the reduced feature space. The proposed deep learning based IoT intrusion detection framework is suitable for FoG layer deployment and can scale well with effective use of sub-divided dataset.

Index Terms—IoT, FoG, Intrusion Detection, Deep Learning, Feature Selection, RNN

I. INTRODUCTION

The wide adoption of Internet of Things (IoT) is paving the way for realizing a highly connected world, which brings numerous benefits to humanity. However this growth has also attracted the attention of adversaries who are constantly searching for ways and methods to penetrate the security perimeters. This is highlighted by a sharp increase in cyber-attacks that target vulnerable smart devices [1]. Such attacks can adversely affect the IoT-enabled networks, especially when they are connected to CI. For example time delays in smart grids, eHealth systems, transportation and manufacturing can significantly affect the service level and safety of CI systems. In order to prevent these intrusions that target IoT devices, superior Intrusion Detection Systems (IDS) are required that do not just rely on attack signatures and instead use traffic characteristics to detect anomalous network connections. To address this problem a number of Deep Learning (DL) based IDS techniques have been proposed [2] which have superior detection performance compared to the traditional Machine Learning (ML) techniques such as SVM [3].

Even though DL techniques provide superior performance, they are computationally intensive and are often deployed using centralised or cloud based architectures [4]. This also results in detection delays as large volumes of data generated by IoT devices need to be pooled to centralised nodes to train the DL models [5], [6]. Hence, such methods are not feasible for intrusion detection in delay sensitive CIs. To counter this, FoG based distributed IDS have been proposed which can effectively detect attacks closer to the edge network and prevent critical delays in detecting malicious activities in IoT devices [6]. FoG nodes can efficiently offload computation tasks of a centralised cloud node resulting in quicker response to cyberattacks and achieve higher scalability with large deployments of IoT devices.

In addition, large datasets would require complex computation and large memory to train deep learning models on FoG nodes [4]. Existing deep learning IDS methods that have been proposed for the FoG layer, either require large number of FoG nodes [6], or move the edge traffic to central nodes to train the DL algorithm [7] which can increase the communication overhead. Other solutions have proposed the use of complex deep learning algorithms such as DL based Long Short-Term Memory Encoder (LAE) to reduce feature dimensionality of the dataset. However such complex feature selection techniques that are based on back propagation algorithm require longer training time and can result in deployment delays. In order to improve the performance for a delay sensitive system we propose a two-step process to simplify the implementation of DL based IDS for FoG nodes. This is achieved by first splitting the time-series based IoT network data according to the attack class where in a multi-class problem is converted to binary class problem. This is followed by applying simple feature reduction technique such as Group Method of Data Handling (GMDH), Mutual Information (MI) and Chi-Square Statistic to reduce the data size for training the DL models. These steps allow the distribution of training tasks to a distributed deployment of FoG nodes as well as reduce computation time and the delay in moving the data across various nodes. This proposed method was evaluated using the BoT-IoT dataset as it contains IoT relevant attack and normal traffic. The main contributions of this work are:

- Split of the BoT-IoT time-series data into smaller data sets according to attack class and time of arrival of packets for distributed processing.
- Adoption of feature selection algorithms that considerably reduce the dataset size required for training by

removing features that do not have strong correlation with the output class without loosing the class differentiation ability.

- Application of Recurrent Neural Networks (Simple-RNN and Bi-directional LSTM) to detect attack traffic in BoT-IoT dataset [8].

The remainder of this paper is structured as follows. Section 2 provides the related work that deal with IoT attack detection and its application on FoG layer. The proposed IDS framework with feature selection step and DL method used in this work are presented in Section 3. In Section 4, the experimental setup, time-series dataset extraction, architecture of the deep learning model along with the obtained results and its analysis is presented. The limitations and future directions of the this study are provided in Section 5, and the conclusion is presented in Section 6.

II. RELATED WORK

In [9], the authors deployed a bi-directional LSTM deep learning algorithm to train and classify sequential network traffic in cloud networks. The proposed approach yielded a high detection rate (between 95-99%) in classifying DoS and DDoS attack traffic when evaluated on UNSW-NB15 and BoT-IoT datasets. However, the results also show that the model performed poorly when classifying non-DoS traffic such as Reconnaissance attacks and information theft.

In our previous work [3], we adopted a Feed forward Neural Netowrk (FNN) to train and classify attacks for a BoT-IoT dataset. The FNN model achieved $> 99\%$ accuracy and a high F1 score in detecting several attack classes. However, the trained FNN model yielded lower precision and recall values for certain categories of IoT attacks.

Authors in [10] proposed a hybrid deep learning method by adopting a single hidden layer Long Short-Term Memory Autoencoder (LAE) layer for dimensionality reduction and cascading the same with a Bi-LSTM layer. The proposed approach resulted in reduced memory utilisation and better performance than other feature reduction techniques. However, the deep learning based feature selection algorithms can incur a high computation time for feature selection.

In [11], the authors proposed an ensemble hybrid IDS comprising an information gain-based feature selection stage, and an ensemble of two shallow ML algorithm, namely, C5 and LIBSVM. Experimental results for the BoT-IoT dataset indicate that individually the shallow classifiers performed poorly but within an ensemble, the performance drastically improved. The C5 and LIBSVM classifiers only achieved 93% and 92% accuracy, respectively, compared to the 99.9% accuracy with ensemble classifiers.

In an another work [12], a deep learning based forensic model called Deep-IFS is proposed, to improve the multi-head attention (MHA) technique that boosts an RNN model in detecting intrusions for Industrial IoT (IIoT) traffic. The proposed model uses a gated RNN unit and an MHA layer to capture both local and global representations of IIoT network traffic. Experiments conducted on FoG nodes indicated that the their proposed approach achieved significant improvement in

performance when compared to centralised DL IDS methods. However, the proposed distributed model requires a large number of FoG nodes to improve the detection accuracy and its performance might worsen with large volumes of traffic.

In summary, the existing literature shows that there are many challenges in implementing deep learning models for intrusion detection at the FoG layer. The main challenges include effectively distributing the detection task to many worker nodes, computation complexity, time delay and bandwidth requirements. Hence in this work we first perform the splitting of multi-class BoT-IoT data set into a binary class taking the time of packet arrival into consideration. Secondly the feature selection is performed on individual binary class datasets by comparing between three dimensionality reduction algorithms to reduce the size of dataset required for training and testing. Finally a SimpleRNN and Bi-LSTM based deep learning models are built which can classify the instance into normal or attack class.

III. PROPOSED FRAMEWORK

In this section, we present the proposed framework for detecting intrusions in IoT traffic. Figure 1 illustrates the various steps of the framework, including obtaining the raw network packets, time-series conversion, feature selection, training, testing and validating the two chosen deep learning methods.

Fig. 1: Workflow of the framework.

The BoT-IoT dataset [8] was adopted in this work to test the performance of the intrusion detection approach when using an RNN. The reason for selecting this dataset was that it contains a wide variety and a significant amount of attack and benign classes related to an IoT deployment. The dataset contains ten subcategories of attack traffic which includes DoS/DDoS attacks over TCP/UDP/HTTP, reconnaissance traffic such as OS fingerprinting and port scanning, data exfiltration and keylogging attacks. The testbed and the collected traffic are explained in detail in [8].

Further analysis of the multi-class BoT-IoT dataset indicates that various attacks were launched in different time periods as highlighted in Table I. Hence instead of performing a mutli-class attack detection, we chose to convert this dataset into a binary classification problem by separating the attack instances into sub-datasets containing only one attack category and normal traffic during that time period arranged according to the packet arrival time.

TABLE I: Date and time analysis of BoT-IoT dataset

Category	Sub-Category	Date	Time (hour)
DDoS	DDoS_HTTP	2018-06-04	09
	DDoS_TCP	2018-06-04	07
	DDoS_UDP	2018-06-04	08
DoS	DoS_HTTP	2018-06-04	05
	DoS_TCP	2018-06-04	03
	DoS_UDP	2018-06-04	04
Data Theft	Data_Exfiltration	2018-06-18	01
	Keylogging	2018-06-19	03,04
Reconnaissance	OS	2018-05-21	04,05,09,10
		2018-05-22	04,05,09,10
		2018-05-15	01-04
	Service	2018-05-21	05-13
		2018-05-22	00-06
Normal	Normal	2018-05-15	00-04
		2018-05-21	05-14
		2018-05-22	00-06,09,10
		2018-06-04	03-10
		2018-06-18	01
		2018-06-19	04

The first step in detecting intrusions is the conversion of the raw network traffic data into packet level features, which is discussed in detail in [3]. We used 29 packet header fields as features including fields in frame, IP, TCP/UDP, and HTTP header. The next step involved converting the packet level data set into a time-series dataset, adopted to train the RNN deep learning algorithm. The *frame.time_epoch* feature was queried to identify the frame arrival time. Once the timestamps were extracted, pre-processing of data was performed, including steps for embedding to encode operation on categorical data. In particular, the column *http.request.method* and port columns were encoded based on various categories present in the dataset. In the next step, individual attack instances were separated and ordered according to packet timestamps to create sub-datasets, which included attack instances of a single class and normal instances as background traffic during the same time period.

Furthermore, features of subdatasets were then evaluated using three feature selection algorithms, namely, Group Method of Data Handling (GMDH), Mutual Information (MI) and Chi-Square statistic methods, to reduce the dimensionality of the datasets.

A. Feature Selection

An important step in building an effective network intrusion detection framework lies in selecting the most appropriate features and removing redundant ones. The feature selection step is applied to reduce the high-dimensional time-series data obtained from the BoT-IoT dataset so that it can be introduced to the machine learning platform to train and test.

1) *GMDH*: GMDH is one of the earliest known deep learning feed-forward networks [13]. It belongs to a heuristic class of algorithms that generate self-organization models automatically with optimal complexity through identification of input feature relationships. The algorithm subsequently self-defines its structure without the need for external intervention. This is achieved by introducing a polynomial known as the Ivakhnenko polynomial [14] to define the relationship between the input variables $x_1, x_2, \dots, x_m$ and the output variable y as:

$$y = a + \sum_{i=1}^m b_i \cdot x_i + \sum_{i=1}^m \sum_{j=1}^m c_{ij} \cdot x_i \cdot x_j + \dots = 1 \quad (1)$$

In equation 1, m is the number of variables included in each neuron layer and a, b, c are weights of these variables. In order to find the best relationship between the input and output variables or to generate models of optimal complexity, a natural evolution process is adopted by the GMDH algorithm [15], which inductively learns the relationships that exist in the data. This is done by deriving complex relationships from simpler forms of equations/relationships, which are introduced as input to the algorithm. The algorithm is provided with $m(m-1)/2$ high-order variables to predict the output y rather than the originally presented m input variables [15]. In addition, variables or features that do not contribute or have high correlation with the output variable are discarded from the process, thus reducing the computation overhead.

The GMDH evaluates all pairwise combinations of input features and selects the best features, referring to the best found relationships between input and output vectors. Steps involved in execution of the GMDH algorithm are enumerated as follows [16]:

- 1) Features are selected in pairwise order and fed to a single neuron.
- 2) At each neuron, the training set is evaluated to estimate the weights.
- 3) At each neuron, probabilities are calculated from the training and validation datasets.
- 4) Using an external criterion, the best neurons are selected. The possible criterion options as provided in the GMDHpy implementation [17] are: validation error, bias error, validation and bias error, bias error first and then on the total dataset (train + validation). The option selected in this work was *validation error*.
- 5) Users can also specify the number of neurons to be selected in each layer or these can also be automatically set based on the number of input variables.
- 6) The algorithm repeats the above steps until a stopping criteria is met such as validation error, maximum layers and if only one neuron gets selected.

The GMDH algorithm adopted for this work performs feature selection and its output feature set are subsequently introduced as inputs to a deep learning implementation. Two sets of reference functions were used while evaluating the GMDH algorithm, namely: linear and linear co-variance (linear_cov). The linear reference function combines the input variables (i.e x_1 and x_2) in a linear combination of two inputs along with their associated weights (w) as shown in 2.

$$y = w_0 + w_1x_1 + w_2x_2 \quad (2)$$

whereas, the linear covariance function includes the two inputs (i.e x_1 and x_2) and the covariation of input variables along with their associated weights (w) as shown in 3.

$$y = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 \quad (3)$$

2) *Mutual Information*: Mutual information (MI) is one of widely used metric to select features based on their goodness measure [18], which is adopted by feature selection algorithms to search the whole feature space, and to select the best possible subset of features. The feature goodness metric depends

on the amount of information that it provides to the output and its independence with respect to other features. MI identifies the level of dependence between two random variables and is based on the principles of Information Theory. Instead of simply identifying the linear relationship between two random variables X and Y , MI provides a measure of information on Y given X . For this to be true, variables X and Y should not be independent. The MI of two random variables X and Y can then be defined as:

$$I(X; Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) * \log\left(\frac{p(x, y)}{p(x)p(y)}\right) \quad (4)$$

where $p(x, y)$ is the joint probability distribution of X and Y . $p(x)$ and $p(y)$ are marginal probability distributions of X and Y . In terms of entropy $H(\cdot)$ it can be defined as:

$$I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X; Y) \quad (5)$$

where $H(X|Y)$ and $H(Y|X)$ are conditional entropies and $H(X; Y)$ is the joint entropy of X and Y . The value of MI is zero if both X and Y are independent and increases in value depending on the levels of dependency between the two variables.

3) *Chi-Squared* (χ^2): χ^2 statistical tests help determine the independence of two events or two attributes. In simple terms, this feature selection technique tests whether the given feature is capable of differentiating the target class attribute by calculating the observed and expected values of a feature as shown in Equation 6.

$$\chi^2 = \sum_{i=1}^n \frac{(O_i - E_i)^2}{E_i} \quad (6)$$

where, O_i is the observed frequency and E_i is the expected frequency of an attribute. In terms of the instances and features of a dataset, the Chi-Square statistic measures the lack of independence between feature f and output class c [19]. Given a network anomaly detection dataset with N instances and two classes: *normal* and *attack*, the importance of a feature x in differentiating the target class can be calculated using the χ^2 statistic using a two-way contingency table represented in Table II, where A is the number of *normal* instances that contain feature x , B is the number of *attack* instances that contain feature x , C is the number of *normal* instances that do not contain feature x , and D is the number of *attack* instances that do not contain feature x .

TABLE II: Chi-Square test contingency table

	Normal Class	Attack Class	Total
Feature x occurs	A	B	A+B
Feature x does not occur	C	D	C + D
Total	A+C	B+D	N

where A, B, C and D are the observed feature values and E_A, E_B, E_C and E_D are the expected values, respectively. The expected values can be calculated as:

$$E_A = (A + C) \frac{A + B}{N} \quad (7)$$

Subsequently, the term goodness-measure [19] can be computed as follows:

$$\chi^2(t, c) = \frac{N \times (AD - CB)^2}{(A + C) \times (B + D) \times (A + B) \times (C + D)} \quad (8)$$

Using a hypothesis evaluation, Chi-Square feature selection selects the most suitable features that have independence scores above a defined threshold. Hence the features that are independent of a target class contain little information in classifying the instance and therefore yield low χ^2 scores.

For MI and Chi-Sqr, the Scikit learn [20] SelectKBest library was deployed to select the top K features that have strong relationship with the output variable. In this work the value of K is set to 10 for a fair comparison with the GMDH method, which selects fewer than 10 features for all classes of the dataset.

4) *Feature Selection Steps*: Experiments conducted to select suitable features for deep learning involved four steps. In the first step, various classes of sub-datasets are generated, followed by extraction of 30% of the dataset for feature ranking and selection. This would reduce the time taken to evaluate the best features as evaluating a full dataset would entail a very high computation time. In the third step, the sampled dataset is fed to GMDH, MI and CHI-Sqr algorithms to select the best features. For the GMDH algorithm, the default split option provided in GMDHpy of 67-33% split was adopted for training and testing purposes. The algorithm by default splits the training data into training and validation. The GMDHpy library was deployed in this work to evaluate the best features from the dataset, under a default configuration. For MI and Chi-Sqr, a min-max scaling was performed to remove any negative values and scale the input values between the range (0,1) across the entire dataset. The MI algorithm in scikit learn uses a nonparametric method that depends on entropy calculations, performed through the k-nearest neighbour method, as highlighted in [21].

B. RNN

In the RNN implementation, each cell accepts one data input and a single hidden state, which propagates with every time step over to the next. The hidden layer activation carries some information from one time step to the next. The main steps involved in a RNN are: calculating the parameters of a single time step and looping over T_n time steps to completely process the input data. A single RNN cell architecture takes x^t as current data input and the h^{t-1} as the previous hidden state, which carries the past information. A single RNN cell architecture is shown in 2.

Hence the current hidden state h^t is calculated as:

$$h^t = G(W_{hh}h^{t-1} + W_{hx}x^t + b_h) \quad (9)$$

where G is the activation function in the hidden layers, W are the weights and b is the bias. The prediction can be obtained as $\hat{y}^t = \text{softmax}(W_{yh}h^t + b_y)$, a sequential combination of all such RNN cells based on the number time steps forms a single RNN layer. The total number of parameters required to train (T_p); the RNN layer can be obtained as: $T_p = [\text{units} * \text{units}]W_{hh} + [\text{input} * \text{units}]W_{hx} + [\text{units}]b_h$

Fig. 2: Single RNN Cell with single input and previous hidden state.

C. Bi-LSTM

LSTM is an improved version of RNN wherein, long-term dependencies are taken into consideration for predicting the output class. Hence LSTM can remember information for a longer time duration enabling them to make better decisions. The problem with RNNs is that they suffer from the vanishing gradient problem which weakens their ability to learn long-term dependencies, as there exists prolonged gap from the time input provided to the time when a decision is taken [22]. LSTM overcomes this drawback by implementing gates which help in passing on the information to any cell as required [23] and retaining contextual information for longer periods. LSTMs are further improved by using a bi-directional hidden layer inputs, which feeds input from both forward and backward directions. [24].

IV. RESULTS AND ANALYSIS

A. Experimental Design

A high performance computing (HPC) cluster with 8 GeForce GTX 1080 Ti GPU running on Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz with 256GB memory was deployed to run the experiments. TensorFlow library and Keras libraries were used for implementing the SimpleRNN and bi-directional LSTM modules. The dataset was split into a 64-16-20 configuration with 64% for training, 16% for validation purposes, and 20% for testing the developed models. This splitting method was based on the Pareto principle which follows the 80/20 rule for splitting the dataset into training and testing. The training set is further divided into 64 and 16 for training and validation respectively. The validation set provides an unbiased evaluation of the trained models and also aids in hyperparameter tuning.

B. Dataset

The BoT-IoT dataset is first converted into a *csv* format comprising only packet level features. The dataset is further converted into a time-series by considering the packet arrival times (*'frame.time_epoch'*). The instances of each attack class are separated in sub-datasets and grouped based on the time period with the inclusion of normal instances in the same period. Next a preprocessing step is carried out where all

the redundant instances are removed along with encoding of categorical variables such as HTTP request methods. Finally, dataset instances are normalised to fit within the range of -1 to 1. The number of instances in each time period belonging to a class and number of features extracted in each class are presented in Table III. For each category of attack and instances observed in its time period, a binary class dataset (normal and attack) was obtained and adopted to carry out feature selection and subsequent training, validation and testing of the deep learning models.

TABLE III: Statistics of the processed dataset.

Category	Normal Class	Attack Class	no. of Features
Service_Scan	1974709	999895	91
OS-Scan	981759	827940	82
Data Ex-filtration	169477	301710	70
Key-logging	124326	11387	75
DDoS-HTTP	152100	194417	73
DDoS-TCP	213432	999444	70
DDoS-UDP	321645	1000055	68
DoS-HTTP	186311	287480	73
DoS-TCP	304006	998703	70
DoS-UDP	391919	999977	68

C. Model Design

The keras implementation of RNN (SimpleRNN) was used in this work to build a model using the top ranked features. The number of parameters for RNN model were based on the attack category chosen as it affects the input shape. For example, for the *service* attack category, there are 91 input features and with a three second window along with 128 neurons, the total number of trainable parameters in the first layer are: $T_p = [128 * 128]W_{hh} + [91 * 128]W_{hx} + [128]b_h = 28160$. After applying the GMDH feature selection algorithm, eight features were selected and hence the number of new trainable parameters in the first layer of RNN reduces to 17536.

Similar to the SimpleRNN, the Bi-directional LSTM model was also built using keras deep learning python library. In both SimpleRNN and Bi-LSTM, a *tanh* activation function in hidden layer was chosen and a *softmax* activation function was selected for the dense layer. Loss value chosen was *sparse_categorical_crossentropy*, with *Adam Optimizer* and a metric chosen was *accuracy*. Figure 3 shows the architecture of RNN model.

The time-series BoT-IoT dataset was first converted into a windowed dataset which selects a fixed number of time samples in the current window and moves sequentially to cover the entire time-series. The *'Window_Size'* parameter controls the number of time samples selected in a given window. The architecture then consists of an input layer which takes the input features. This is followed by a hidden layer consisting of several neurons that performs the computation, and the output layer performs the classification of instances into normal and attack categories. During the training phase, interconnecting weights in the three layers of the deep learning network were calculated and further tuned until an optimal solution was reached. This is done using the back propagation algorithm which updates and chooses an optimal interconnection weights that produce the least loss.

Fig. 3: RNN model architecture.

D. Model Tuning

Hyperparameter tuning: Various hyperparameters associated with RNN were explored to identify the optimal settings for the model tuning. The parameters that were considered for tuning were hidden layers, number of neurons, dropout rate, learning rate, number of epochs, batch size and window size. Results indicate that increasing the hidden layers for SimpleRNN increases the model performance but beyond three hidden layers the performance did not improve; hence the number of hidden layers were set as three in this work. The number of neurons was set to 128 as it provided the best results with all the classes. The dropout rate did not have any effect on the model performance and hence no further tuning was performed. The learning rate of Adam optimizer showed considerable impact on the performance with learning rate as 0.0001 giving the best results hence that value was chosen for model training. Increasing the number of epochs didn't have any impact on the model performance as most iterations ended before 20 epochs. In addition, an early stopping condition was applied with a patience of 5 iterations in order to stop the training if validation loss stays constant or increases with consecutive iterations, this will prevent any over-fitting problem. When dealing with large datasets the batch size effects the stability and speed of the learning process. The evaluation showed that for SimpleRNN a batch size of 128 improved the accuracy of the model. The window size is used to create windowed dataset; the window size of 3 showed the best performance among the values between 2 and 5. In the case of Bi-LSTM the same set of hyperparameter values were chosen as they showed increased performance of the model. However the batch size was increased to 50 which improved the performance of the model. In summary, the hyperparameter values chosen in this work are: *window_size* = 3, *batch_size* = 128, *epoch_time_SimpleRNN* = 20, *epoch_time_BiLSTM* = 50, learning rate = 0.0001, and number of neurons as 128. In addition, three hidden layers with one dense layer were chosen for SimpleRNN and two hidden layers with a single dense layer was selected for Bi-LSTM.

E. Analysis of Results

The evaluation results presented in this section describe the various features selected by the feature selection algorithms

and then the performance of SimpleRNN and bi-directional LSTM models. For evaluating the feature selection results, the features selected by individual feature selection algorithm are presented and the amount of data size reduction occurred with the best selected features is presented. For evaluating the network traffic classification, the confusion matrices are used as a measure for model performance in classifying normal and attack classes in each sub-dataset. Four additional metrics are used for measuring the performance which are: accuracy, recall, precision, and F1 score. In particular, accuracy is the number of instances with correct class predictions among all the instance predictions; recall is the number of true positives compared to the actual number of positive class; precision is the number of true positives compared to the total number of predicted positive class values; F1 score is the weighted average of recall and precision. A low recall value represents a high number of attack instances missclassified as normal while a low value of precision represents a high proportion of false positives where by normal instances are marked as attack instances.

The feature selection using the three algorithms resulted in a considerable amount of data size reduction as highlighted in Table IV. The maximum data size reduction in terms of MegaBytes (MB) occurred with the *service scan* sub-category. In this sub-category, the reduced data required 107,128,128 MB of storage space respectively for GMDH, MI and Chi-Sqr compared to the 997 MB of storage space required for the full dataset. In terms of percentage reduction GMDH method achieved above 90% reduction of data size for data theft, keylogging, DDoS-HTTP, and DoS-HTTP. For MI and Chi-Sqr, 10 features were selected which the amount of size reduction between 80-90%. These results indicate that chosen feature selection methods can effectively reduce the amount of data required to train and evaluate the deep learning models by removing irrelevant and uncorrelated features.

TABLE IV: Percentage memory reduction in with the three feature selection algorithms under each subcategory

Category	GMDH-lr-cov	GMDH-lr	MI	Chi-Sqr
Service	89.24	89.24	87.09	87.09
OS	89.28	86.90	85.71	85.71
Data Exfiltration	90.27	90.27	83.33	83.33
Keylogging	90.90	93.50	84.41	84.41
DDoS-HTTP	90.66	93.33	83.99	83.99
DDoS-TCP	87.49	86.11	83.33	83.33
DDoS-UDP	82.85	84.28	82.85	82.85
DoS-HTTP	91.99	93.33	83.99	83.99
DoS-TCP	87.49	87.49	83.33	83.33
DoS-UDP	84.28	89.99	82.85	82.85
Average	88.45	89.44	84.09	84.09

Using the best selected features by the various algorithms, performance of SimpleRNN and Bi-LSTM models with full features and selected feature sets was compared and presented in the following section.

The experiments conducted on the various subsets of attack traffic and binary classification performed on the instances using SimpleRNN show that the models using best selected features performed either better than the full feature set in most categories or equivalent to the full feature based model. The models based on best feature set also achieved higher recall rates compared to full feature set in most attack sub-categories

Fig. 4: Comparison of recall rate of proposed SimpleRNN and Bi-LSTM based attack detection technique on BoT-IoT dataset with other work.

as listed in Table V. However, in few attack categories such as OS Fingerprinting, Keylogging and DoS-HTTP, the models trained on features selected by GMDH algorithm showed lower precision compared to other models.

A similar observation is recorded for Bi-LSTM models based on full features and with best feature sets. The results indicate that performance of models built using selected features provides improved performance metrics compared to the model built using the full features as listed in Table V. The training and validation loss for Bi-LSTM shows that for most sub-datasets the training and validation process extended beyond 20 epochs but ended before 50 epochs. The training loss reduced below 0.2 for all attack sub-categories after the third epoch. In addition, the validation loss reduced below 0.1 for all sub-categories but fluctuated between 0 and 0.05 for DoS_HTTP and DoS_UDP. The Bi-LSTM model also had overall better performance compared to the SimpleRNN models as it has the ability to consider long term time dependencies while taking decisions.

F. Comparison and Discussion

Comparing the results with work proposed in [9], our Bi-LSTM, SimpleRNN and top performing feature selected models outperform in detecting all attack categories. Figure 4 compares the recall rate of various attack sub-categories with results presented in Alkadi et. al [9]. Our proposed approach showed higher recall rates especially in service scan, OS fingerprint, data ex-filtration and keylogging. The results of feature selection and deep learning based classification show that performing a feature selection process improves the performance of deep learning models in detecting IoT based attacks. The main advantage of applying a feature selection step with deep learning is that it considerably reduces the dataset size without losing the important class-discriminative information between the input and output variables. Among the various feature selection algorithms, features selected by MI algorithm showed highest performance improvement in more than one sub-category.

This can be beneficial in circumstances where a attack detection needs to be distributed into smaller computing nodes such as those deployed in the FoG layer and also increase the generalisation performance. This can considerably reduce

the number of worker FoG nodes and computation resources required for intrusion detection compared to those reported in [12]. Applying a feature selection step before the deep learning layer increases the model interpretability [25], this can greatly enhance the way IDS can be developed due to the understanding of most important features relevant in attack detection.

Limitations: One of the drawbacks of a reduced dataset (only selected features) is that it resulted in a lower precision value for certain categories compared to the full dataset. This limitation can be overcome by creating an ensemble of classifiers trained on top selected features to increase the overall precision values.

V. CONCLUSION

Deep learning techniques applied in intrusion detection systems have shown to accurately identify attack patterns. With increasing attacks targeting the IoT paradigm, such techniques are suitable in detecting these intrusions, especially at the FoG layer. In this work we proposed a IDS framework for the IoT, which can be effectively implemented at the FoG layer. The proposed framework implements a dataset splitting step according to the attack traffic arrival time. Furthermore a feature selection step removes irrelevant features from the high-dimensional BoT-IoT dataset. The reduced dataset is then used to train and test the effectiveness of two RNN algorithms (SimpleRNN and Bi-LSTM) to classify the instance in attack and normal traffic. The obtained results show the efficacy of the proposed framework. The feature selection step reduced the storage space by 90% without losing the class differentiation ability. This was highlighted by the enhanced recall rate of SimpleRNN and Bi-LSTM models with reduced feature space compared to the full feature set as well as Bi-LSTM model presented in [9]. The dataset splitting method, the reduction of storage requirements combined with superior detection capability of deep learning RNN models makes the proposed method scalable and suitable for FoG layer intrusion detection.

REFERENCES

- [1] I. Stelios, P. Kotzanikolaou, M. Psarakis, C. Alcaraz, and J. Lopez, "A survey of iot-enabled cyberattacks: Assessing attack paths to critical infrastructures and services," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3453–3495, 2018.
- [2] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, "A survey of deep learning-based network anomaly detection," *Cluster Computing*, pp. 1–13, 2019.
- [3] M. Ge, X. Fu, N. Syed, Z. Baig, G. Teo, and A. Robles-Kelly, "Deep learning-based intrusion detection for iot networks," in *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2019, pp. 256–25609.
- [4] L. Lyu, J. C. Bezdek, X. He, and J. Jin, "Fog-embedded deep learning for the internet of things," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4206–4215, 2019.
- [5] S. Otoum, B. Kantarci, and H. T. Mouftah, "On the feasibility of deep learning in sensor network intrusion detection," *IEEE Networking Letters*, vol. 1, no. 2, pp. 68–71, 2019.
- [6] A. Diro and N. Chilamkurti, "Leveraging lstm networks for attack detection in fog-to-things communications," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 124–130, 2018.
- [7] A. Samy, H. Yu, and H. Zhang, "Fog-based attack detection framework for internet of things using deep learning," *IEEE Access*, vol. 8, pp. 74 571–74 585, 2020.

TABLE V: Performance comparison based on features selection and attack category for SimpleRNN and BiLSTM

Attack Category	Feature selection	#Features	SimpleRNN - Performance				Bi-LSTM - Performance			
			Accuracy	F1	Precision	Recall	Accuracy	F1	Precision	Recall
Service	No-Feature selection	91	0.9919	0.9748	0.9732	0.9764	0.9901	0.9692	0.9701	0.9684
	GMDH-lr-cov	8	0.9801	0.9396	0.9191	0.961	0.9818	0.9438	0.9392	0.9483
	GMDH-lr	8	0.9921	0.976	0.9592	0.9934	0.9928	0.9781	0.9621	0.9947
	MI	10	0.9921	0.976	0.9603	0.9922	0.9928	0.9781	0.9616	0.9953
	CHI-SQR	10	0.9914	0.9737	0.96	0.9879	0.9758	0.929	0.8765	0.9883
OS Fingerprint	No-Feature selection	82	0.9809	0.9527	0.9147	0.9941	0.993	0.9822	0.9716	0.993
	GMDH-lr-cov	7	0.9785	0.9469	0.9082	0.989	0.9832	0.9582	0.9264	0.9922
	GMDH-lr	9	0.9393	0.8632	0.7644	0.9913	0.9494	0.8832	0.797	0.9904
	MI	10	0.9848	0.9619	0.9313	0.9947	0.993	0.9823	0.9681	0.9968
	CHI-SQR	10	0.9839	0.9598	0.9276	0.9944	0.9952	0.9878	0.9791	0.9967
Data Exfiltration	No-Feature selection	70	0.9946	0.9709	0.9531	0.9893	0.999	0.9944	0.9937	0.9951
	GMDH-lr-cov	5	0.9325	0.7258	0.5716	0.9942	0.9973	0.9852	0.9736	0.9971
	GMDH-lr	5	0.9972	0.9847	0.9736	0.9961	0.9973	0.9852	0.9736	0.9971
	MI	10	0.995	0.9731	0.9528	0.9942	0.999	0.9949	0.9932	0.9966
	CHI-SQR	10	0.9948	0.9722	0.9498	0.9956	0.9948	0.9722	0.949	0.9966
Keylogging	No-Feature selection	75	0.9965	0.9704	0.9515	0.9901	0.9972	0.9761	0.9656	0.9868
	GMDH-lr-cov	5	0.9947	0.9549	0.9461	0.9638	0.9947	0.9557	0.9304	0.9824
	GMDH-lr	3	0.9933	0.9435	0.9251	0.9627	0.9928	0.9412	0.9024	0.9835
	MI	10	0.9955	0.9627	0.9349	0.9923	0.9957	0.9644	0.9342	0.9967
	CHI-SQR	10	0.9956	0.9637	0.9368	0.9923	0.9956	0.9639	0.9332	0.9967
DDoS - HTTP	No-Feature selection	73	0.9977	0.9884	0.9817	0.9952	0.998	0.9898	0.9818	0.9981
	GMDH-lr-cov	5	0.9942	0.97	0.9811	0.9592	0.995	0.9742	0.9812	0.9673
	GMDH-lr	3	0.9465	0.6679	0.8405	0.5542	0.9755	0.8664	0.9233	0.8162
	MI	10	0.9979	0.9896	0.9818	0.9976	0.9981	0.9903	0.9818	0.999
	CHI-SQR	10	0.9979	0.9896	0.9818	0.9976	0.9991	0.9957	0.992	0.9995
DDoS - TCP	No-Feature selection	70	0.9999	0.9999	1	0.9998	0.9999	0.9999	1	0.9998
	GMDH-lr-cov	7	0.9999	0.9999	1	0.9999	0.9999	0.9999	0.9999	0.9999
	GMDH-lr	8	0.9999	0.9999	1	0.9999	0.9999	0.9999	1	0.9999
	MI	10	0.9997	0.9998	0.9999	0.9997	0.9995	0.9997	1	0.9994
	CHI-SQR	10	0.997	0.9981	0.9998	0.9964	0.9994	0.9996	0.9996	0.9996
DDoS - UDP	No-Feature selection	68	0.9954	0.9969	0.9995	0.9942	0.9979	0.9985	0.9992	0.9978
	GMDH-lr-cov	10	0.9954	0.9969	0.9984	0.9953	0.998	0.9987	0.9982	0.9991
	GMDH-lr	9	0.9799	0.9863	0.9937	0.979	0.9961	0.9973	0.9997	0.995
	MI	10	0.9952	0.9967	0.9995	0.9939	0.9991	0.9994	0.9996	0.9991
	CHI-SQR	10	0.995	0.9966	0.9967	0.9965	0.9967	0.9978	0.9968	0.9987
DoS - HTTP	No-Feature selection	73	0.9973	0.973	0.9481	0.9992	0.9987	0.9868	0.9777	0.996
	GMDH-lr-cov	4	0.989	0.884	0.8907	0.8775	0.989	0.8968	0.8129	1
	GMDH-lr	9	0.9929	0.9297	0.8878	0.9758	0.994	0.9377	0.9295	0.9461
	MI	10	0.9973	0.973	0.9475	1	0.9973	0.973	0.9475	1
	CHI-SQR	10	0.9973	0.973	0.9475	1	0.9983	0.9827	0.966	1
DoS - TCP	No-Feature selection	70	0.9999	0.9999	0.9999	0.9999	1	1	1	1
	GMDH-lr-cov	7	0.9999	0.9999	1	0.9999	0.9999	0.9999	1	0.9999
	GMDH-lr	7	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999
	MI	10	0.9968	0.9979	0.9999	0.9958	0.9997	0.9998	0.9999	0.9997
	CHI-SQR	10	0.9979	0.9986	0.9996	0.9976	0.9948	0.9965	0.9994	0.9935
DoS - UDP	No-Feature selection	68	0.9968	0.998	0.9995	0.9964	0.9987	0.9992	0.9992	0.9997
	GMDH-lr-cov	9	0.9976	0.9985	0.9995	0.9974	0.999	0.9993	0.9997	0.999
	GMDH-lr	5	0.9757	0.9844	0.9991	0.9701	0.8779	0.9165	0.9999	0.8459
	MI	10	0.9939	0.9961	0.9994	0.9929	0.9964	0.9977	0.9995	0.9959
	CHI-SQR	10	0.9819	0.9884	0.9989	0.9782	0.9983	0.9989	0.9996	0.9982

- [8] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset," *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019.
- [9] O. Alkadi, N. Moustafa, B. Turnbull, and K.-K. R. Choo, "A deep blockchain framework-enabled collaborative intrusion detection for protecting iot and cloud networks," *IEEE Internet of Things Journal*, 2020.
- [10] S. I. Popoola, B. Adebisi, M. Hammoudeh, G. Gui, and H. Gacani, "Hybrid deep learning for botnet attack detection in the internet of things networks," *IEEE Internet of Things Journal*, 2020.
- [11] A. Khraisat, I. Gondal, P. Vamplew, J. Kamruzzaman, and A. Alazab, "A novel ensemble of hybrid intrusion detection system for detecting internet of things attacks," *Electronics*, vol. 8, no. 11, p. 1210, 2019.
- [12] M. Abdel-Basset, V. Chang, H. Hawash, R. K. Chakraborty, and M. Ryan, "Deep-ifs: Intrusion detection approach for iiot traffic in fog environment," *IEEE Transactions on Industrial Informatics*, 2020.
- [13] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0893608014002135>
- [14] A. G. Ivakhnenko, "Polynomial theory of complex systems," *IEEE transactions on Systems, Man, and Cybernetics*, no. 4, pp. 364–378, 1971.
- [15] S. J. Farlow, *Self-organizing methods in modeling: GMDH type algorithms*. CrC Press, 2020.
- [16] O. Dag, E. Karabulut, and R. Alpar, "Gmdh2: Binary classification via gmdh-type neural network algorithms—r package and web-based tool," *International Journal of Computational Intelligence Systems*, vol. 12, pp. 649–660, 2019. [Online]. Available: <https://doi.org/10.2991/ijcis.d.190618.001>
- [17] kvoyager, "Gmdhpy," 2017. [Online]. Available: <https://github.com/kvoyager/GmdhPy>
- [18] F. Amiri, M. R. Yousefi, C. Lucas, A. Shakery, and N. Yazdani, "Mutual information-based feature selection for intrusion detection systems," *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1184–1199, 2011.
- [19] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *Proceedings of the Fourteenth International Conference on Machine Learning*, ser. ICML '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, p. 412–420.
- [20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
- [21] B. C. Ross, "Mutual information between discrete and continuous data sets," *PloS one*, vol. 9, no. 2, p. e87357, 2014.
- [22] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 8624–8628.
- [23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [24] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.
- [25] J. Chen, L. Song, M. J. Wainwright, and M. I. Jordan, "Learning to explain: An information-theoretic perspective on model interpretation,"

APPENDIX

ADDITIONAL PERFORMANCE RESULTS

Fig. 5: Training Loss with Bi-LSTM for all categories of attack datasets

Fig. 6: Validation Loss with Bi-LSTM for all categories of attack datasets