Hindawi
Wireless Communications and Mobile Computing
Volume 2020, Article ID 6689134, 16 pages
https://doi.org/10.1155/2020/6689134

Research Article

Intrusion Detection System for Internet of Things Based on Temporal Convolution Neural Network and Efficient Feature Engineering

Abdelouahid Derhab[1], **Arwa Aldweesh**[2], **Ahmed Z. Emam**[2], **and Farrukh Aslam Khan**[1]

[1]*Center of Excellence in Information Assurance (CoEIA), King Saud University, Saudi Arabia*
[2]*College of Computer and Information Sciences (CCIS), King Saud University, Saudi Arabia*

Correspondence should be addressed to Abdelouahid Derhab; abderhab@ksu.edu.sa

Received 13 October 2020; Revised 23 November 2020; Accepted 4 December 2020; Published 23 December 2020

Academic Editor: Xiaojie Wang

In the era of the Internet of Things (IoT), connected objects produce an enormous amount of data traffic that feed big data analytics, which could be used in discovering unseen patterns and identifying anomalous traffic. In this paper, we identify five key design principles that should be considered when developing a deep learning-based intrusion detection system (IDS) for the IoT. Based on these principles, we design and implement Temporal Convolution Neural Network (TCNN), a deep learning framework for intrusion detection systems in IoT, which combines Convolution Neural Network (CNN) with causal convolution. TCNN is combined with Synthetic Minority Oversampling Technique-Nominal Continuous (SMOTE-NC) to handle unbalanced dataset. It is also combined with efficient feature engineering techniques, which consist of feature space reduction and feature transformation. TCNN is evaluated on Bot-IoT dataset and compared with two common machine learning algorithms, i.e., Logistic Regression (LR) and Random Forest (RF), and two deep learning techniques, i.e., LSTM and CNN. Experimental results show that TCNN achieves a good trade-off between effectiveness and efficiency. It outperforms the state-of-the-art deep learning IDSs that are tested on Bot-IoT dataset and records an accuracy of 99.9986% for multiclass traffic detection, and shows a very close performance to CNN with respect to the training time.

1. Introduction

The Internet of Things (IoT) network is a set of smart devices such as sensors, home appliances, phones, vehicles, and computers that are interconnected through the global Internet. This type of network is increasingly becoming an essential part of our everyday life and is providing a variety of applications such as smart home, smart grid, smart agriculture, smart cities, and intelligent transportation.

Although the IoT can make the human's life more comfortable, this benefit comes at the expense of security [1]. Nowadays, IoT networks are becoming an attractive target for cybercriminals and are exposed to major risks. A report from Unit 42 of Palo Alto Networks revealed that 98% of all IoT device traffic is unencrypted, and 41% of attacks exploit IoT device vulnerabilities [2]. The vulnerable devices could be later used by adversaries to join an IoT botnet and participate in sophisticated and large-scale attacks. For example, the first IoT botnet launched in October 2016, named Mirai [3], was able to compromise vulnerable CCTV cameras that were using default usernames and passwords to launch a DDoS attack on DNS servers. This attack resulted in stopping the Internet accessibility in some parts of the USA. In April 2020, an IoT botnet, named Mozi, was discovered and was found capable of launching various DDoS attacks [4, 5].

To deal with this kind of threat, the intrusion detection systems have been widely used to detect malicious network traffic [6, 7], especially when the preventive techniques fail

at the level of endpoint IoT devices. As cyberattacks targeting IoT are increasingly becoming more sophisticated and stealthy, the IDS should continuously evolve to handle emerging security threats. Due to its heterogeneous nature, IoT network generates high-dimensional, multimodal, and temporal data. By applying big data analytics on such data, it is possible to discover unseen patterns, reveal hidden correlations, and gain new insights [8]. Artificial intelligence is increasingly used in the big data analysis process. In particular, deep learning techniques have proven their success in dealing with heterogeneous data [8–11]. They are also capable of analyzing complex and large-scale data to get insights, spot dependencies within data, and learn from previous attack patterns to recognize new and unseen attack patterns [12–14]. As IoT devices are resource-constrained and have limited capabilities in terms of storage and computation, heavyweight tasks like big data analysis process and building of learning models need to be offloaded to fog and cloud servers [15–21]. Hence, computation offloading [22] can help reduce the execution delay of the task and save energy consumption of battery-powered and mobile IoT devices, but it also poses some security concerns [23].

Many deep learning approaches have been proposed for IDS, and some of them specifically focus on IoT [24–32]. Each approach adopts its own design choices, which might limit its capability in achieving good performance in terms of effectiveness and efficiency.

In this paper, we propose five design principles to be considered when developing an effective and efficient deep learning IDS for IoT, and we use these principles to propose TCNN, a variant of CNN that uses causal convolutions. TCNN is combined with data balancing and efficient feature engineering. More specifically, the main contributions of the paper are the following:

(i) We identify five key design principles for the development of deep learning-based IDS for IoT, including *handling overfitting*, *balancing dataset*, *feature engineering*, *model optimization*, and *testing on IoT dataset*

(ii) Based on the identified key design principles, we compare the state-of-the art methods, identify their gaps, and analyze the main differences with respect to our work.

(iii) We design and implement Temporal Convolution Neural Network (TCNN), a deep learning framework for intrusion detection systems in IoT. TCNN combines Convolution Neural Network (CNN) with causal convolution.

(iv) To handle the issue of imbalanced dataset, we integrate TCNN with Synthetic Minority Oversampling Technique-Nominal Continuous (SMOTE-NC).

(v) We employ efficient feature engineering, which consists of the following:

(1) *Feature space reduction*: it helps in reducing memory consumption.

(2) *Feature transformation*: it is applied on continuous numerical features using *log transformation* and *standard scaler*, which transforms skewed data to Gaussian-like distribution. It is also applied on categorical features using *label-encoding*, which replaces a categorical column with a unique integer value.

(vi) We evaluate the effectiveness and efficiency of the proposed TCNN on Bot-IoT dataset, and compare it with CNN, LSTM, logistic regression, random forest, and other state-of-the-art methods. The results show the superiority of TCNN in scoring an accuracy of 99.9986% for multiclass traffic detection.

The rest of the paper is organized as follows. Section 3 presents the key design principles with respect to deep learning IDS for IoTs. Section 4 overviews related work. Section 4 and Section 5 describe the design and implementation of TCNN, respectively. Section 6 presents the evaluation results and comparison with state-of-the-art methods. Finally, Section 7 concludes the paper and outlines future research directions.

2. Key Design Principles for Deep Learning IDS in IoT

The objective of deep learning-based IDS solutions for IoT is to generate models that perform well in terms of effectiveness and efficiency. However, each model adopts some design choices that might limit its ability in achieving this objective. For example, some deep learning IDSs in IoT do not consider the overfitting problem, or apply their model on an unbalanced dataset, or neglect employing feature engineering, which negatively affects their performance in terms of accuracy, memory consumption, and computational time. Also, some IDSs do not try to optimize their learning model, and some are evaluated on outdated or irrelevant datasets, which do not reflect real-world IoT network traffic.

Motivated by the above observations, the deep learning-based IDS solution for IoT should advocate the following key design principles:

(i) *Handling overfitting*: overfitting happens when the model achieves a good fit on the training data, but it does not generalize well on unseen data. In deep learning, overfitting could be avoided by the following methods:

(1) Applying regularization, which adds a cost to the loss function of the model for large weights.

(2) Using dropout layers, which randomly remove certain features by setting them to 0.

(ii) *Balancing dataset*: data imbalance refers to a disproportion distribution of classes within a dataset. If a model is trained under an imbalanced dataset, it will become biased, i.e., it will favor the majority classes

and fail to detect the minority classes. By balancing the dataset, the effectiveness of the model will be improved.

(iii) *Feature engineering*: it allows reducing the cost of the deep learning workflow in terms of memory consumption and time. It also allows improving the accuracy of the model by discarding irrelevant features and applying feature transformation to improve the accuracy of the learning model.

(iv) *Model optimization*: the objective of model optimization is to minimize a loss function, which computes the difference between the predicted output and the actual output. This is achieved by iteratively adjusting the weights of the model. By applying an optimization algorithm such as SGD and Adam [33], the effectiveness of the model will be improved.

(v) *Testing on IoT dataset*: a deep learning-based IDS for IoT should be tested under an IoT dataset to get results that reflect real-world IoT traffic.

3. Related Work

Deep learning has been applied in many fields of cybersecurity including malware detection [34–39] and intrusion detection system [14, 40–46]. In this section, we give an overview on deep learning-based IDS for IoT networks.

Lopez et al. [26] proposed RNN-CNN, a combination of recurrent neural network (RNN) and convolutional neural network (CNN). To deal with overfitting, they added some layers such as max pooling, batch normalization, and dropout. They only considered a subset of features to improve the effectiveness of the model.

Putchala [28] applied the Gated Recurrent Unit (GRU) algorithm on KDD 99Cup dataset. He also used the random forest classifier as a feature selection technique. The best possible performance results are obtained by minimizing the loss function.

Roy and Cheung [31] presented Bidirectional Long Short-Term Memory Recurrent Neural Network (BLSTM RNN). They applied feature normalization and converted categorical features to numeric values.

Diro et al. [24] applied a deep neural network (DNN) on NSL-KDD dataset. The loss function of DNN is minimized using stochastic gradient descent (SGD). There are fog nodes, which are responsible for training the deep learning model. The local parameters are sent to a fog coordinator node for update. This allows sharing the best parameters and helps avoiding local overfitting.

Roopak et al. [29] applied four different classification deep learning models: MLP, 1d-CNN, LSTM, and CNN +LSTM on CICIDS2017 dataset. They also balanced the dataset by duplicating the records. However, it is not explained which balancing method is used. The overfitting issue is handled by adding some layers to the model such as max pooling and dropout.

In [32], Deep Belief Network (DBN) is used to develop a feed-forward deep neural network (DNN) and is applied on an IoT simulation dataset. DNN is optimized by assigning a cost function to each layer of the model.

Otoum et al. [27] proposed Stacked-Deep Polynomial Network (SDPN) on NSL-KDD dataset. For optimal selection of features, they employed the Spider Monkey Optimization (SMO) algorithm [47]. To avoid overfitting, the L2 regularization technique is integrated with the loss function.

Ferrag and Maglaras [25] applied recurrent neural network (RNN) with the truncated backpropagation through time (BPTT) algorithm on two non-IoT datasets and BoT-IoT dataset. They normalized the features before feeding them to RNN-BPTT.

Roopak et al. [30] proposed a sequential architecture combining CNN and LSTM and applied it on CISIDS2017 dataset. For optimal selection of features, they employed a multiobjective optimization algorithm named nondominated sorting genetic algorithm (NSGA) [48]. To avoid overfitting, they implemented a max-pooling layer between CNN and LSTM layers.

Koroniotis et al. [49] are the first who developed BoT-IoT dataset, and they used it to test RNN and LSTM. For feature selection, they computed the correlation coefficient among the features of the dataset and applied feature normalization to scale the data within the range [0, 1].

Aldhaheri et al. [50] proposed DeepDCA, an IDS that combines Dendritic Cell Algorithm (DCA) and Self Normalizing Neural Network (SNN). They adopted Information Gain as a feature selection technique to decide on the set of features to be fed to BoT-IoT dataset. Although, the authors presented results with balanced dataset but no information about balancing method is provided. As for model optimization, they used a loss function to update the weights of the deep learning layers.

Soe et al. [51] proposed Artificial Neural Network (ANN) to detect DDoS attacks in Bot-IoT dataset. To balance the dataset, they used the SMOTE technique. Also, they applied feature normalization before feeding the input data to ANN.

Ge et al. [52] applied feed-forward neural networks (FNN) on BoT-IoT dataset. The dataset is balanced not through oversampling but in an algorithmic way, i.e., giving class weights to the training data. To optimize the model, they used Adam optimizer and a sparse categorical cross-entropy loss function to update weights. To deal with overfitting, they employed different regularization techniques such as L1, L2, and dropout. They also encoded categorical features as numerical using one-hot encoding.

Muna et al. [53] proposed a combination of deep autoencoder (DAE) and deep feed-forward neural network (DFFNN) to detect malicious activities in industrial IoT. The optimal parameters are obtained by calculating a loss function, which allows updating the weights and minimizes the difference between the actual and the predicted output.

3.1. Key Finding. Table 1 summarizes and compares the IDS solutions with respect to the abovementioned five design principles. We can notice that only 6 out of 14 solutions are tested under IoT dataset [25, 27, 49–51]. The majority of solutions do not consider dataset balancing. Only 4 solutions are designed with data balancing [29, 50–52], two of them do

Table 1: Deep learning-based IDS for IoT.

Ref	DL technique	Overfitting	Unbalanced dataset	Feature engineering	Model optimization	Testing on IoT dataset
[26]	CNN-RNN	Yes	No	FS	No	No: RedIRIS
[28]	GRU	No	No	FS:RF	Yes	No: KDDCup'99
[31]	BLSTM RNN	No	No	FE	No	No: UNSW-NB15
[24]	DNN	Yes	No	FE	Yes	No: NSL-KDD
[29]	MLP, 1d-CNN, LSTM, CNN+LSTM	Yes	Yes	No	No	No: CICIDS2017
[32]	DNN	No	No	No	Yes	Yes: simulation
[27]	SDPN	Yes	No	FS:SMO	Yes	No: NSL-KDD
[25]	RNN-BPTT	No	No	FN	No	Yes: Bot-IoT
[30]	CNN+LSTM	Yes	No	FS: NSGA	No	No: CISIDS2017
[49]	RNN, LSTM	No	No	FS:CC	No	Yes: Bot-IoT
[3]	DeepDCA	No	Yes	FS:IG	Yes	Yes: BoT-IoT
[51]	ANN	No	SMOTE	FN	No	Yes: BoT-IoT
[52]	FNN	Yes	Yes	FE	Yes	Yes: BoT-IoT
[53]	DAE-DFFFN	No	No	FE, FN	Yes	No: NSL-KDD, UNSW-NB15
Our work	TCNN	Yes	SMOTE-NC	FSR FT: LT, SS, FE	Yes	Yes: BoT-IoT

FS: feature selection; RF: random forest; FE: feature encoding; FN: feature normalization; FSR: feature space reduction; IG: information gain; CC: correlation coefficient; FT: feature transformation; LT: log transformation; SS: standard scaler; SMO: Spider Monkey Optimization; NSGA: nondominated sorting genetic algorithm.

not explain how the balancing approach is implemented [29, 50], one solution considers algorithmic-level data balancing [52], and only one solution considers data-level balancing by applying SMOTE algorithm [51]. Handling overfitting is not considered in the design of 7 solutions [25, 28, 31, 32, 49, 50, 53]. On the other hand, model optimization is only considered by 7 solutions [24, 27, 28, 32, 50, 52, 53]. Most of the solutions employ feature engineering in their design, except for two solutions [29, 32].

3.2. Comparison with Related Work. To the best of our knowledge, our wok and [52] are the only ones that consider all the five design principles. Differently from [52], which adopts algorithmic-level data balancing, our work applies the SMOTE-NC algorithm on Bot-IoT dataset, which can handle continuous and categorical features. We use overfitting and optimization techniques in achieving effective IDS. We also use feature space reduction and feature transformation in achieving efficient and lightweight IDS in terms of memory usage and training time.

4. Proposed Framework

4.1. Basic Principles. Deep learning is a concatenation of different layers. The first layer is called the input layer, and the last layer is called the output layer. In addition, hidden layers are inserted between the input and the output layers. Each layer is composed of a set of units, called neurons. The size of the input layer depends on the dimension of the input data, whereas the output layer is composed of C units, which corresponds to the C classes of a classification task.

Convolutional neural network (CNN), as shown in Figure 1, is a deep neural network that is composed of multiple layers. The three main types of layers are the following:

(i) *Convolutional layer*: it applies a set of filters, also known as convolutional kernels, on the input data. Each filter slides over the input data to produce a feature map. By stacking all the produced feature maps together, we get the final output of the convolution layer

(ii) *Pooling layer*: it operates over the feature maps to perform subsampling, which reduces the dimensionality of the feature maps. Average pooling and max pooling are the most common pooling methods

(iii) *Fully connected layer*: It takes the output of the previous layers, and turns them into a single vector that can be an input for the next layer

The TCNN deep learning architecture [54] is a combination of CNN architecture and causal padding, which results in causal convolutions. Figure 2 shows 1D causal convolution with a kernel size of 3, which is applied on time-series input data $(x_0, x_1, \cdots, x_T)$. By causal convolutions, we mean that an output at time t is convolved only with elements from time t and earlier in the previous layer. Therefore, it does not violate the temporal order of the data, and there is no leakage of information from future to past. Zero padding of length $(\text{kernel size} - 1)$ is added to the layers to have the same length as the input causal convolution layer.

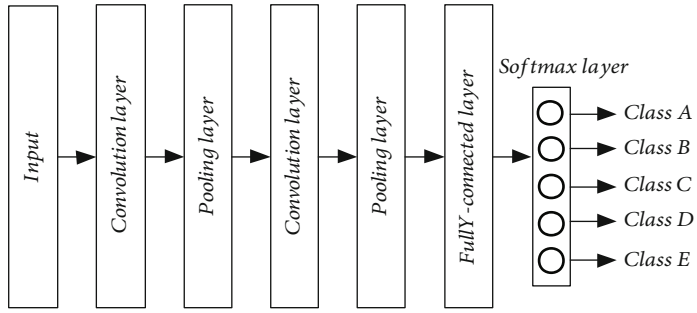


Figure 1: CNN architecture.

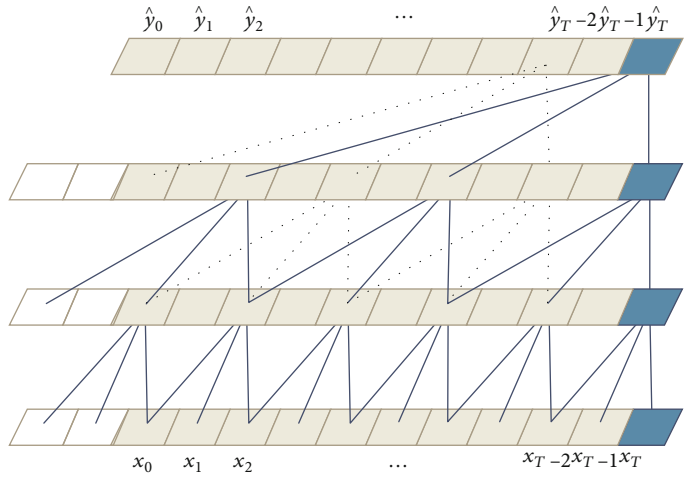


Figure 2: 1D causal convolution [54].

4.2. Overall Architecture. Figure 3 shows the overall architecture of the proposed TCNN framework, and its implementation is detailed in Section 5. The proposed architecture is composed of the following phases:

(i) *Dataset balancing*: as mentioned above, an imbalanced dataset can produce misleading results. To handle this problem, we use in this phase the SMOTE-NC method, which creates synthetic samples of minority classes and is capable of handling mixed dataset of categorical and continuous features.

(ii) *First feature engineering (feature space reduction)*: in this phase, we clean the dataset, i.e., reduce the feature space by removing unnecessary features, and converting the memory-consumption features into lower-size datatype.

(iii) *Dataset splitting*: in this phase, the dataset is split into: training, validation, and testing subsets in order to counter overfitting.

(iv) *Second feature engineering (feature transformation)*: in this phase, we apply feature transformation on the training subset. Log transformation and standard scaler are applied on the continuous numerical features. In addition, label encoding is applied to categorical features, which simply replaces each categorical column with a specific number. This transformation process is later applied on the validation and the testing subsets.

(v) *Training and optimization*: in this phase, the TCNN model is built, as described in Section 4.3. It is trained using the training subset, and its parameters are optimized using Adam optimizer and the validation subset.

(vi) *Classification:* the generated TCNN model is applied on the testing subset to attribute each testing record to its actual class: normal or a specific category of attack.

4.3. Training and Optimization of TCNN Framework. The training and optimization phase of the proposed TCNN is composed of two 1D causal convolution layers, two dense layers, and a softmax layer, which applies softmax functions for multiclass classification task. To overcome overfitting, we use global maximum pooling, batch normalization, and dropout layers. We choose Adam optimizer to update weights and optimize cross-entropy loss function. Adam optimizer combines the advantages of two stochastic gradient descent algorithms, namely Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp).

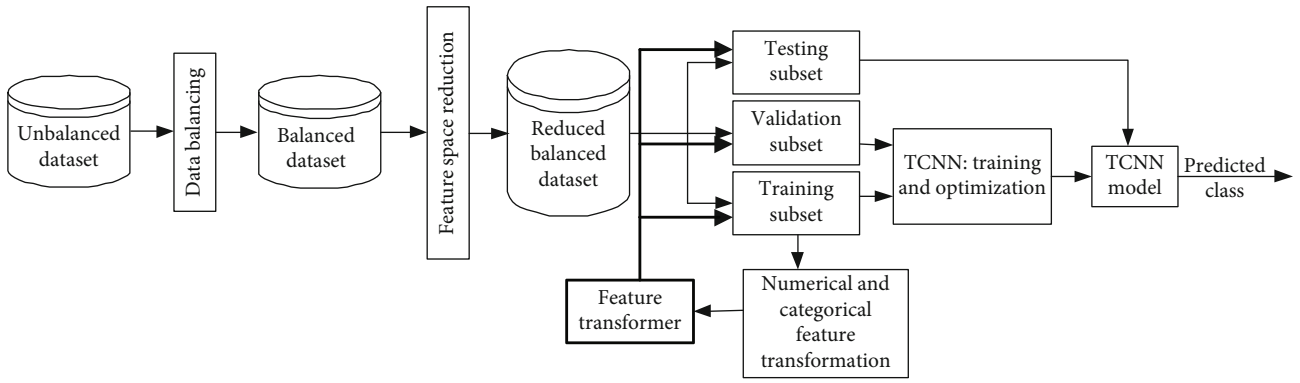


Figure 3: TCNN framework.

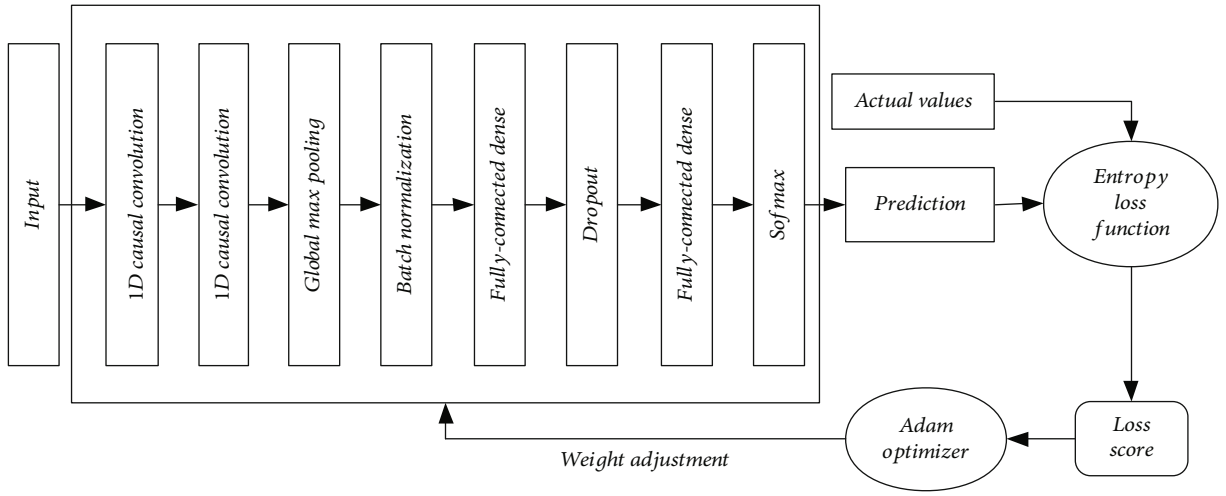


Figure 4: Training and optimization of the proposed TCNN framework.

Specifically, the training and optimization phase of the proposed TCNN architecture, as shown in Figure 4, is composed of the following layers:

(i) *First 1D causal convolution layer*: it convolves across the input vectors with 64 filters and filter size of 3.

(ii) *Second 1D causal convolution layer*: it uses 128 filters and a filter size of 3. This second layer before pooling allows the model to learn more complex features.

(iii) *1D global maximum pooling layer*: it replaces data, which is covered by the filter, with its maximum value. It prevents overfitting of the learned features by taking the maximum value.

(iv) *Batch normalization layer*: it normalizes the data coming from the previous layer before going to the next layer.

(v) *Fully connected dense layer*: it employs 128 hidden units and a dropout ratio of 30%.

(vi) *Fully connected dense layer with softmax activation function*: it produces five units that correspond to the five categories of traffic for multiclass classification.

5. Implementation

To implement the detection learning models, we use Intel Quad-core i7-8550U processor with 8 GB RAM and 256 GB Hard drive. As for software, we use Python 3.6 programming language, and TensorFlow to build deep learning models. Moreover, different libraries are used including Scikit-learn, Keras API, Panda, and Inmblearn. We implement the framework in Figure 3 on Bot-IoT Dataset [49].

5.1. Bot-IoT Dataset. We use Bot-IoT [49], an IoT dataset that was released in 2018 by the Cyber Center in the University of New South Wales. By virtualizing the setup of various smart home appliances including weather stations, smart fridges, motion-activated lights, remotely activated garage doors, and smart thermostats, legitimate and malicious traffic is generated. The dataset consists of more than 73,000,000 records, which are represented by 42 features, as shown in Table 2. Each record is labeled either as normal or attack. In addition, the attack dataset is divided into four categories: DoS, DDoS, reconnaissance, and information theft, and each category is further divided into subcategories, as shown in Table 3.

Table 2: Features of Bot-IoT dataset.

Feature	Description	Data type	Feature	Description	Data type
pkSeqID	Row identifier	Integer	Dpkts	Destination-to-source packet count	Integer
stime	Record start time	Float	Sbytes	Source-to-destination byte count	Integer
flgs	Flow state flags seen in transactions	Category	Dbytes	Destination-to-source byte count	Integer
proto	Textual representation of transaction protocols presents in network flow	Category	Rate	Total packets per second in transaction	Float
Saddr	Source IP address	Category	Srate	Source-to-destination packets per second	Float
Sport	Source port number	Category	Drate	Destination-to-source packets per second	Float
Daddr	Destination IP address	Category	TnBPSrcIP	Total number of bytes per source IP	Integer
Dport	Destination port number	Category	TnBPDstIP	Total number of bytes per destination IP.	Integer
Pkts	Total count of packets in transaction	Integer	TnP_PSrcIP	Total number of packets per source IP.	Integer
Bytes	Total number of bytes in transaction	Integer	TnP_PDstIP	Total number of packets per destination IP.	Integer
State	Transaction state	Category	TnP_PerProto	Total number of packets per protocol.	Integer
Ltime	Record last time	Float	TnP_Per_Dport	Total number of packets per dport	Integer
Seq	Argus sequence number	Integer	AR_P_Proto_P_SrcIP	Average rate per protocol per source IP. (calculated by pkts/dur)	Float
Dur	Record total duration	Float	AR_P_Proto_P_DstIP	Average rate per protocol per destination IP.	Float
Mean	Average duration of aggregated records	Float	N_IN_Conn_P_SrcIP	Number of inbound connections per source IP.	Integer
Stddev	Standard deviation of aggregated records	Float	N_IN_Conn_P_DstIP	Number of inbound connections per destination IP.	Integer
Sum	Total duration of aggregated records	Float	AR_P_Proto_P_Sport	Average rate per protocol per sport	Float
Min	Minimum duration of aggregated records	Float	AR_P_Proto_P_Dport	Average rate per protocol per dport	Float
Max	Maximum duration of aggregated records	Float	Pkts_P_State_P_Protocol_P_SrcIP	Number of packets grouped by state of flows and protocols per source IP.	Integer
Spkts	Source-to-destination packet count	Integer	Pkts_P_State_P_Protocol_P_DestIP	Number of packets grouped by state of flows and protocols per destination IP	Integer

Table 3: Bot-IoT dataset statistics.

Normal/attack	Category	Subcategory	Number of records
Attack	Reconnaissance (2.48%)	Service scanning	1,463,364
		OS fingerprinting	358,275
	DoS (52.25%)	TCP	19,547,603
		UDP	18,965,106
		HTTP	19,771
	DDoS (44.98%)	TCP	12,315,997
		UDP	20,659,491
		HTTP	29,706
	Information theft (0.22%)	Keylogging	1,469
		Data exfiltration	118
Normal (0.13%)			9,543
	Total		73,370,443

In this work, we use a subset of Bot-IoT dataset, consisting of approximately 3,700,000 records, which is the same as the one used in [49].

5.2. Dataset Balancing. In the dataset, there are 9,543 normal and 73,360,900 attack samples. The subset of the dataset is composed of 477 normal samples and 3,668,045 attack samples. We can notice that more than 97% of the samples belong to DoS and DDoS categories, as shown in Table 3. In this way, the learning model will predict the majority classes and fail to spot the minority classes, which means the model is biased.

To deal with this problem, different resampling methods have been proposed [55] like (1) random oversampling, which randomly replicates the exact samples of the minority classes, and (2) oversampling by creating synthetic samples of minority classes using techniques such as synthetic minority oversampling technique (SMOTE), synthetic minority oversampling technique for nominal and continuous (SMOTE-NC), and adaptive synthetic (ADASYN). In this work, we use the SMOTE-NC technique as it is capable of handling mixed dataset of categorical and continuous features [56]. The minority classes such as normal and theft are increased to 100,0000 samples in the training subset, as shown in Table 4.

5.3. Feature Space Reduction. One of the main objectives of this work is to develop a lightweight IDS for IoT environment. Therefore, it is important to improve the efficiency of the detection models by reducing the feature space and noise in the dataset, as well as reducing the memory usage and computation complexity. By using the full set of features, 2.9 GB of memory is used. Feature space reduction decreases the processing complexity and speeds up the training and detection processes. The following steps are applied to the dataset, which successfully decrease the memory consumption to 668 MB, i.e., 77% reduction.

Table 4: Training dataset: left: original dataset, and right: oversampling dataset.

DDoS	1541299	DDoS	1541299
DoS	1320208	DoS	1320208
Reconnaissance	72865	Normal	100000
Normal	382	Theft	100000
Theft	63	Reconnaissance	72865

Table 5: Data type of features.

Data type	Int64	Float64	Object
# features	22	15	9

(i) *Conversion of object data type into categorical data type*: Table 5 shows the data types and the number of features encoded for each type. As shown in the table, there are 9 memory-consuming features that are encoded as objects, which are "flgs," "proto," "saddr," "sport," "daddr," "dport," "state," "category," and "subcategory." As category datatype is more efficient, object features are converted into category datatype [57].

(ii) *Conversion of Int64 data type into Int32 data type*: by default, the 22 integer features in the dataset, as shown in Table 2, are stored as Int64 (8-bytes) type. After checking these features, we find out that they do not exceed the capacity of Int32 (4-bytes) type. Therefore, all the values of Int64 type are encoded into Int32 type, which incurs half of the memory consumption that is incurred by the Int64 type.

(iii) *Removing unnecessary features*: in the dataset, we exclude some useless features such as the following:

(1) "pkSeqID": it has the same role as the automatically generated index.

(2) "stime" and "ltime": they are captured in the "*dur*" feature, which computes the duration between "*stime*" and "*ltime*".

5.4. Feature Transformation. We describe how the numerical features and categorical features are transformed. After the dataset is split into training, validation, and testing subsets, the transformation is only applied on the training subset. Then, the same transformation is reapplied on the validation and the testing subsets.

(1) *Numerical feature transformation*: the dataset contains 31 numerical features, including both discrete and continuous values. There are two discrete features, i.e., "*spkts*" and "*dpkts*," and are represented by a finite number of values. So, they do not require any feature engineering.

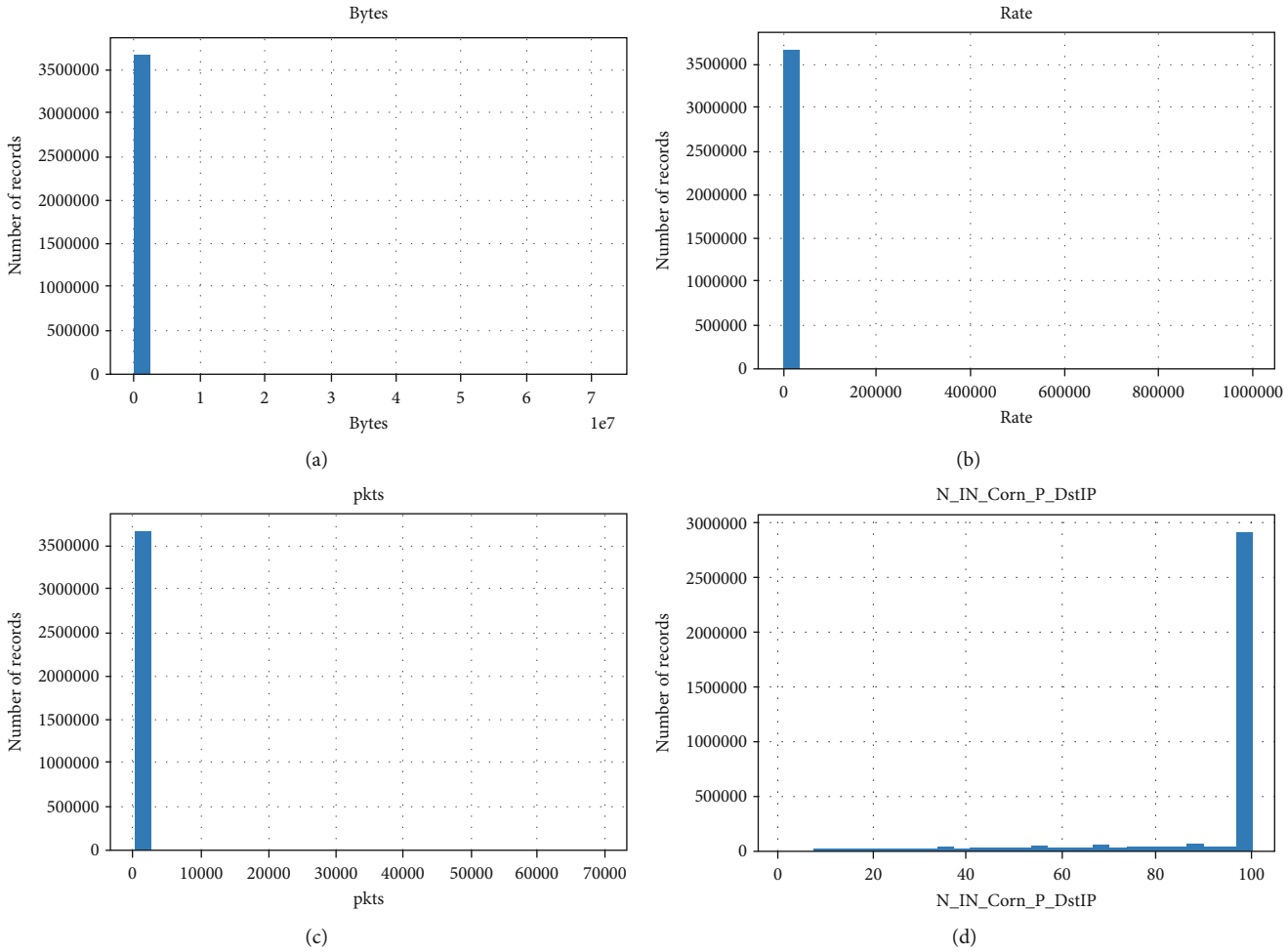


Figure 5: Histogram of some continuous feature before transformation.

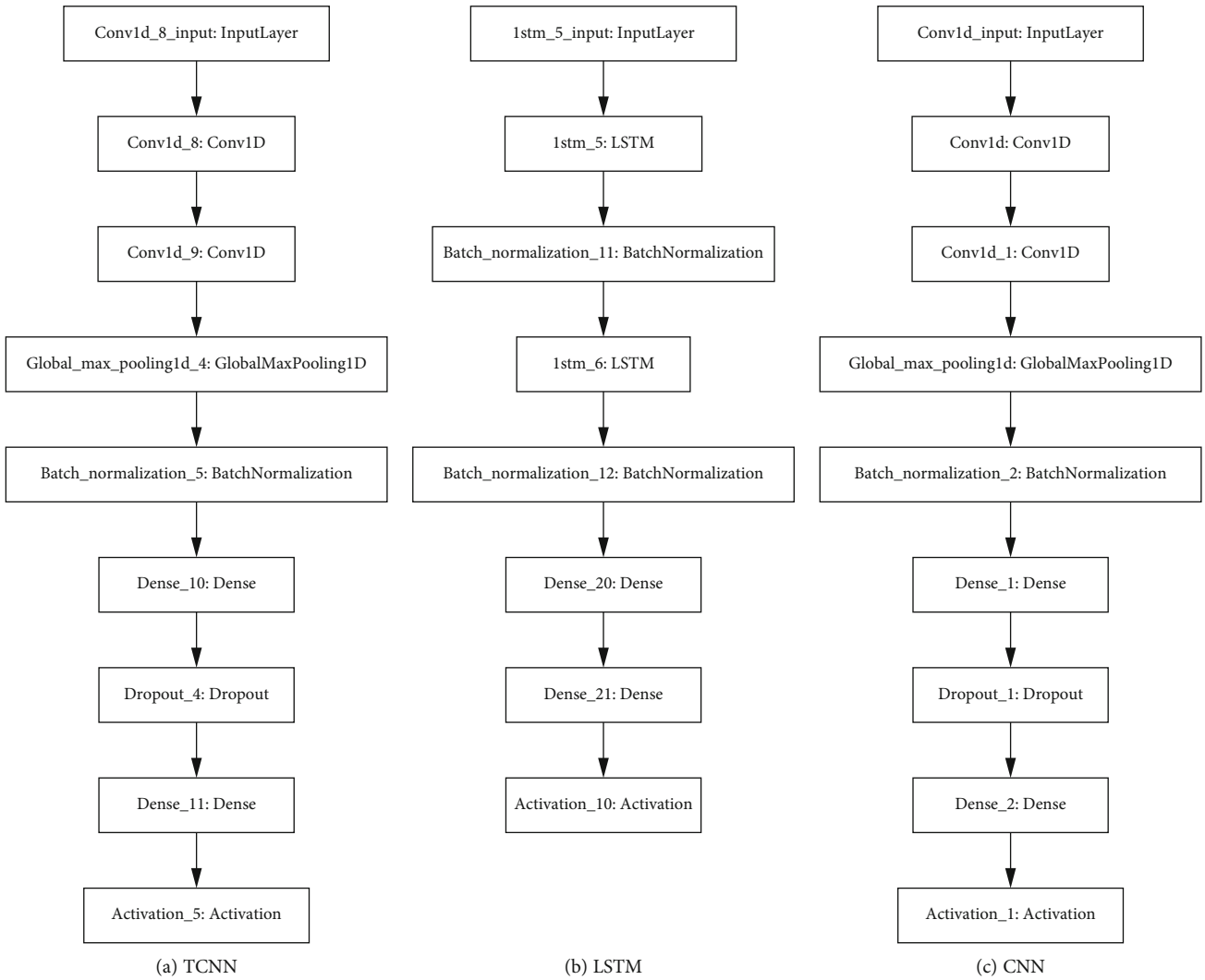
There are 29 continuous features in the dataset, which are "pkts," "bytes," seq, dur,, mean, stddev, sum, min, max, spkts, dpkts, sbytes, dbytes, rate, srate, drate, TnBPSrcIP, TnBPDstIP, TnP_PSrcIP, TnP_PDstIP, TnP_PerProto, TnP_Per_Dport, AR_P_Proto_P_SrcIP, AR_P_Proto_P_DstIP, N_IN_Conn_P_DstIP, N_IN_Conn_P_SrcIP, AR_P_Proto_P_Sport, AR_P_Proto_P_Dport, Pkts_P_State_P_Protocol_P_DestIP, and Pkts_P_State_P_Protocol_P_SrcIP. Figure 5 shows the histograms of 4 features. As shown in the figure, the continuous features are not normally distributed, which usually affects the performance of linear models. To this end, log transformation and standard scaler are applied to the continuous features to be Gaussian-like distribution as follows:

(i) *Log transformation*: the new value of the feature $x' = \log_{10} x$, where x is the original value of the feature.

(ii) *Standard scaler*: it computes the mean μ and standard deviation σ on a training set. Then, the features are normalized to Gaussian distribution. For each x', we compute the normalized value $x'' = x' - \mu/\sigma$

5.5. Dataset Splitting. Conventional splitting and cross-validation are the main approaches used to split datasets. Cross-validation is mainly used in legacy machine learning to overcome the overfitting problem. When a large dataset is used with deep learning, cross-validation increases the training cost. In this work, the dataset is split using the conventional three-way split into: training, validation, and testing subsets. In addition, regularization is applied to deal with the overfitting if it appears [58]. Also, a stratified split is used to ensure that there is a portion of each class in each split [59].

5.6. Deep Learning Models. All deep learning models are built using Keras API on top of TensorFlow. Different Keras packages are used, including preprocessing, models, layers, optimizers, and callback. The same activation functions are used in all models. To model nonlinear relationships between input and output in each layer, relu activation function is used. The output layer activation function is softmax; a generalized logistic regression activation function is used. The number of output units in softmax is equivalent to the number of attack categories in addition to the normal class [60]. The deep learning architectures of TCNN, LSTM, and CNN are shown in Figure 6, and their hyperparameters are shown in Table 6.



(a) TCNN (b) LSTM (c) CNN

Figure 6: Deep learning models.

To deal with overfitting, some techniques such as Global maximum pooling, Batch normalization, and dropout are used. To adjust the weights, Adam optimizer is selected since it outperforms the other optimizers, such as SGD and AdaGrad.

6. Evaluation

We evaluate the performance of TCNN and compare it with two legacy machine learning algorithms, i.e., logistic regression (LR) and random forest (RF), and two deep learning models, i.e., LSTM, and CNN.

6.1. Performance Metrics. The multiclass detection models are evaluated with respect to the following metrics:

(i) *Effectiveness metrics*: we measure how the detection model is effective in distinguishing between the different classes of network traffic. To this end, we use the following metrics:

(i) Accuracy = TP + TN/TP + TN + FP + FN

(ii) Precision = TP/TP + FP

(iii) Recall = TP/TP + FN

(iv) $F1 - \text{score} = 2(\text{Precision} \times \text{Recall})/\text{Precision} + \text{Recall}$

where TP, TN, FP, and FN denote the true positives, true negatives, false positives, and false negatives, respectively.

(ii) *Log loss (cross-entropy loss)*: it measures the performance of the classification model whose output is a probability value. A perfect model would have a log loss of 0, and it increases as the predicted probability diverges from the actual label. Formally,

$$\text{logloss} = -\frac{1}{C}\sum_{i=1}^{C}(y_i \log p_i + (1 - y_i) \log (1 - p_i)) \quad (1)$$

where C is the number of classes.

Table 6: Hyperparameters of deep learning models.

Hyperparameters		Value	Activation function
TCNN	First 1D causal convolution layer	#filters = 64, filter size = 3	ReLU
	Second 1D causal convolution layer	#filters = 128, filter size = 3	ReLU
	Fully connected dense layer	#neurons = 128, dropout = 0. 3	ReLU
	Fully connected dense layer	#neurons = 5	Softmax
CNN	First 1D convolution layer	#filters = 64, filter size = 3	ReLU
	Second 1D convolution layer	#filters = 128, filter size = 3	ReLU
	Fully connected dense layer	#neurons = 128, dropout = 0. 3	ReLU
	Fully connected dense layer	#neurons = 5	Softmax
LSTM	First LSTM layer	#neurons = 20, recurrent dropout = 0.2	ReLU
	Second LSTM layer	#neurons = 20, recurrent dropout = 0.2	ReLU
	Fully connected dense layer	#neurons = 128	ReLU
	Fully connected dense layer	#neurons = 5	Softmax
Optimizer		Adam with learning rate = 0.001	/
Batch size		1024	/
Epochs		15	/

Table 7: Performance of machine learning models.

Detection model	Oversampling	Phase	Log loss	Accuracy	Precision	Recall	F1-score	Training time (s)
LR	None	Training	0.055841	97.0861%	59.8890%	81.2382%	52.1265%	511
		Testing	0.057109	97.0598%	59.9419%	82.6940%	52.1741%	
	SMOTE-NC	Training	0.075336	99.2955%	75.2781%	99.2496%	79.6344%	709
		Testing	0.077694	99.2858%	74.5496%	98.6987%	78.9640%	
RF	None	Training	0.200992	97.4837%	80.4852%	98.8858%	86.8911%	191
		Testing	0.20116	97.4586%	77.8298%	98.8643%	84.4592%	
	SMOTE-NC	Training	0.195178	96.6396%	79.8083%	98.5854%	86.3145%	197
		Testing	0.195124	96.6341%	75.8464%	98.5543%	82.6850%	

(iii) *Training time*: it measures the required time to build the classification model

6.2. Evaluation of Legacy Learning Models. Logistic regression and random forest are evaluated under original and rebalanced datasets, and their results are shown in Table 7. Training and testing scores are almost similar for all the experiments, which confirm the absence of overfitting. As for logistic regression, SMOTE-NC oversampling leads to an improvement in precision, recall, and F1-score, which means that there is improvement in detecting minority classes. On the other hand, the oversampling does not improve the effectiveness of random forest.

6.3. Evaluation of Deep Learning Models. We conduct a series of experiments with different hyperparameter values (e.g., learning rate, batch size, number of layers, and number of units in each layer) in order to get the best performance. Different learning rates of the optimizer are tested. The best performance is achieved when the learning rate is 0.001. Also, different number of epochs 10, 15, 20, 50, and 100 and different batch sizes 100, 256, 512, and 1024 are tested. We can notice that increasing the number of epochs will slow down the learning process. Similarly, using a smaller batch size does not improve the performance. The number of epochs and the batch size for TCNN are set to 15 and 1024, respectively.

Figure 7 shows the accuracy and log loss of TCNN for multiclass classification during the training and validation phases. TCNN reaches high performance in the first epochs, which emphasizes that 15 epochs would be enough. Additionally, the training and validation results show the absence of overfitting. The log loss results of LSTM and CNN are shown in Figure 8. We can observe that TCNN outperforms LSTM and CNN in terms of log loss.

Tables 8–10 show the performance of TCNN, LSTM, and CNN, respectively. We can observe that deep learning models perform better than LR and RF, as some accuracy results exceed 99.99%. The accuracy results are very close but TCNN slightly outperforms LSTM and CNN in terms of effectiveness metrics. We can also observe that the deep learning models show good results even without applying dataset balancing. By applying, SMOTE-NC oversampling, we record an insignificant and very slight decrease in the effectiveness of TCNN and LSTM. On the other hand, the effectiveness of CNN slightly increases after applying

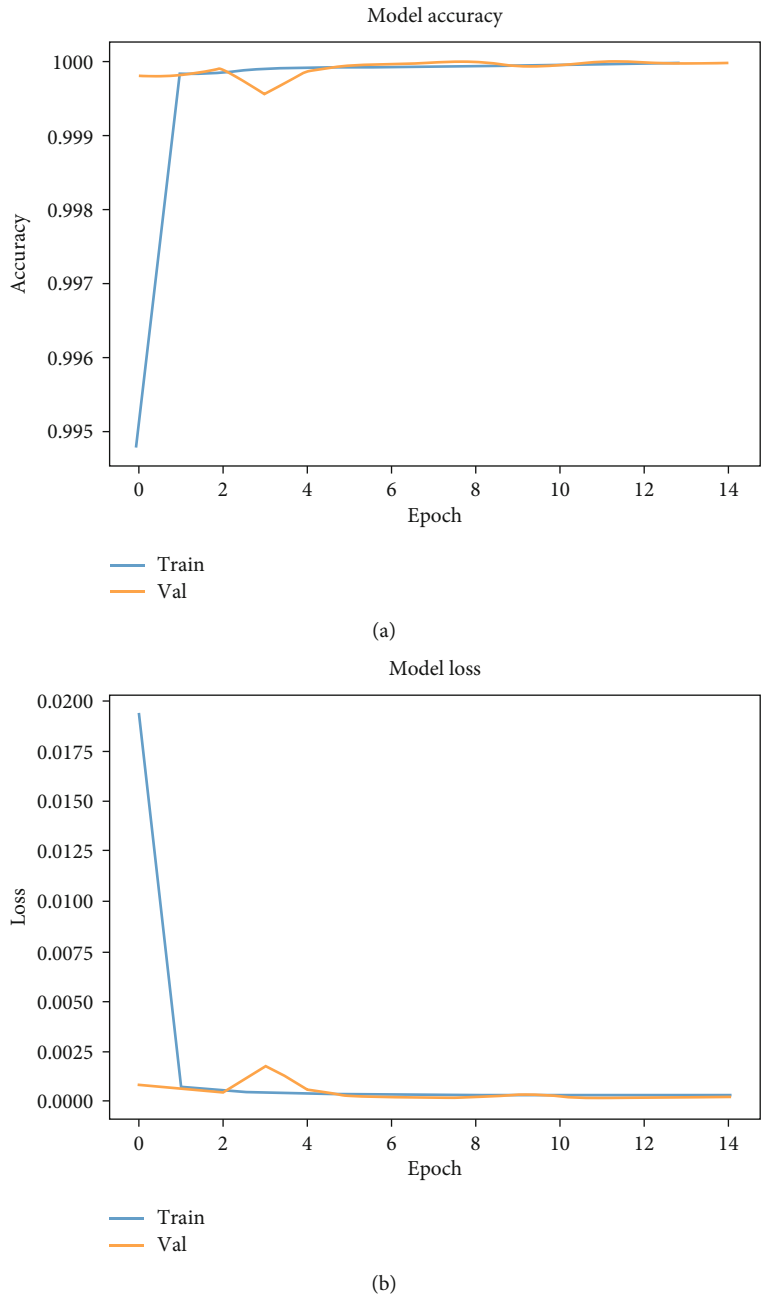


Figure 7: Accuracy and log loss of TCNN vs. number of epochs.

SMOTE-NC oversampling. CNN also incurs lower training time compared to TCNN and LSTM. TCNN offers a good trade-off between effectiveness and efficiency, as it is the closest competitor to CNN with respect to training time, and it records the best accuracy result.

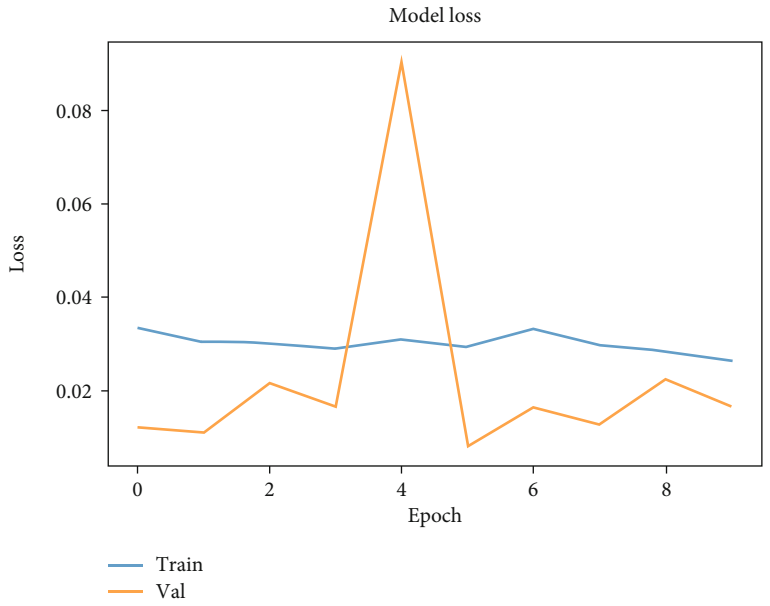
6.4. Comparison with Related Work Tested under Bot-IoT Dataset. In Table 11, we compare the performance of our work with other state-of-the-art methods that are tested under Bot-IoT dataset. The comparison is conducted with respect to accuracy, precision, recall, F1-score, training time, and classification task. According to the table, we can identify the following classification tasks:

(i) *Binary classification task*: it aims to distinguish between normal and attack records.

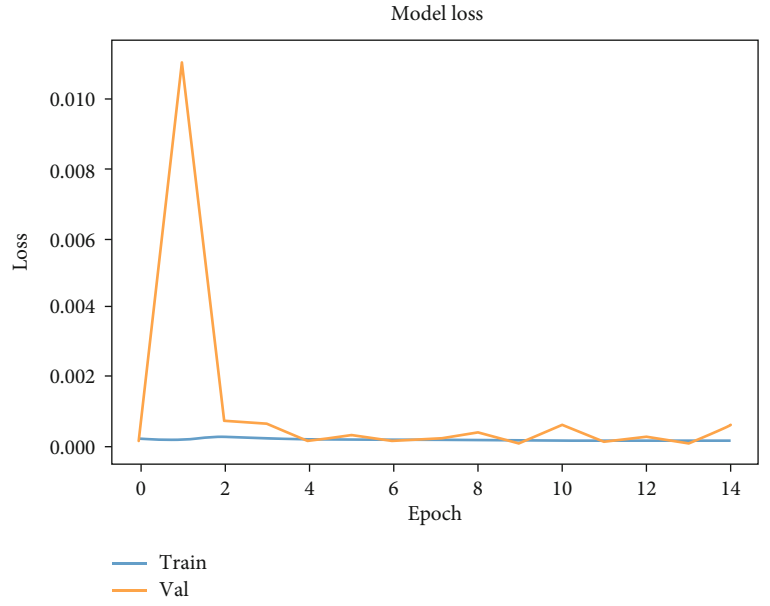
(ii) *Normal/one-attack classification task*: it aims to distinguish between normal records and one type of attacks.

(iii) *Multiclass classification*: it aims to attribute a record to its correct class among the five classes, i.e., one normal class and four attack classes.

It is known that multiclass classification is the most challenging task, whereas the normal/one-attack classification is the easiest one as the dataset only contains one type of attack, which means less diversity within the dataset, and easy



(a) LSTM



(b) CNN

Figure 8: Log loss of LSTM and TCN vs. number of epochs.

Table 8: Performance of TCNN model.

Oversampling	Log loss	Accuracy	Precision	Recall	F1-score	Training time (s)
None	0.000072	99.9986%	99.9974%	97.4975%	98.6641%	424
SMOTE-NC	0.000101	99.9978%	97.1379%	94.9972%	95.9961%	447

Table 9: Performance of LSTM model.

Oversampling	Log loss	Accuracy	Precision	Recall	F1-score	Training time (s)
None	0.002027	99.9654%	99.9443%	84.5703%	89.3016%	762
SMOTE-NC	0.002131	99.9643%	84.0246%	99.5169%	88.5303%	746

Table 10: Performance of CNN model.

Oversampling	Log loss	Accuracy	Precision	Recall	F1-score	Training time (s)
None	0.000094	99.9973%	95.1360%	97.0783%	96.0500%	419
SMOTE-NC	0.000118	99.9984%	99.9952%	94.9975%	97.1392%	490

Table 11: Comparison with related work tested on Bot-IoT dataset.

Ref	Model	Task	Accuracy	Precision	Recall	F1-score	Training time (s)
[49]	RNN	Binary	99.7404%	99.9904%	99.7499%	—	8035
	LSTM	Binary	99.7419%	99.9910%	99.7508%	—	10482.19
[61]	Ensemble learning	Binary	99.97%	—	—	—	—
[25]	RNN with BPTT	Multiclass	99.912%	—	—	—	2012
[50]	DeepDCA	Multiclass	98.73%	99.17%	98.36%	98.77%	—
[51]	ANN	Normal/DDoS	100%	100%	100%	100%	—
[52]	FNN	Multiclass	99.02%	—	—	—	—
Our	TCNN	Multiclass	99.9986%	99.9974%	97.4975%	98.6641%	424
	LSTM		99.9654%	99.9443%	84.5703%	89.3016%	762
Work	CNN		99.9973%	95.1360%	97.0783%	96.0500%	419
	LR		99.2858%	74.5496%	98.6987%	78.9640%	709
	RF		97.4586%	77.8298%	98.8643%	84.4592%	191

learning for the detection model. From Table 11, we can observe that [51] achieves 100% effectiveness. However, this result can be explained by the fact that [51] aims to distinguish between normal traffic and only one type of attack, i.e., DDoS. The three deep learning models, TCNN, LSTM, and CNN, outperform the rest of related work, although they are evaluated under multiclass classification task. We can also observe that TCNN, LSTM, and CNN incur the best results in terms of training time. This is due to the adopted feature engineering that reduces the computation complexity and due to the use of simple deep learning architectures with larger batch size and less number of layers.

7. Conclusion and Future Work

In this paper, we have identified five design principles for the development of an effective and efficient deep learning-based intrusion detection system for the Internet of Things (IoT). By adopting these principles, we have designed and implemented Temporal Convolution Neural Network (TCNN), which combines Convolution Neural Network (CNN) and causal convolution. TCNN is integrated with SMOTE-NC data balancing and efficient feature engineering, which consists of feature space reduction and feature transformation.

TCNN has been evaluated on Bot-IoT dataset and compared with logistic regression, random forest, LSTM, and CNN. Evaluation results show that TCNN achieves a good trade-off between effectiveness and efficiency. It outperforms the state-of-the-art deep learning IDS methods, which were tested under Bot-IoT dataset, by recording an accuracy of 99.9986% for multiclass traffic detection. Also, it shows a very close performance to CNN with respect to training time. As part of future work, it would be interesting to consider another design principle, i.e., testing the resiliency of IDS against adversarial attacks, which can confuse the deep learning model to produce wrong predictions.

Data Availability

We used the Bot-IoT, which is a publicly accessed dataset (https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/bot_iot.php), for the evaluation of the proposed IDS.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors extend their appreciation to the Deanship of Scientific Research at King Saud University for funding this work through research group No (RG-1439-021).

References

[1] H. Belkhiri, A. Messai, M. Belaoued, and F. Haider, "Security in the internet of things: recent challenges and solutions," in *International Conference on Electrical Engineering and Control Applications*, pp. 1133–1145, Constantine, Algeria, 2019.

[2] Palo alto networks, "2020 unit 42 iot threat report," 2020, https://unit42.paloaltonetworks.com/iot-threat-report-2020/.

[3] M. Antonakakis, T. April, M. Bailey et al., "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security17)*, pp. 1093–1110, Vancouver, BC, Canada, 2017.

[4] S. Fadilpasic, "Researchers discover iot botnet capable of launching various ddos attacks," April 2020, https://www.itproportal.com/news/researchers-discover-iot-botnet-capable-of-launching-various-ddos-attacks/.

[5] J. Vijayan, "New malware family assembles iot botnet," April 2020, https://www.darkreading.com/iot/new-malware-family-assembles-iot-botnet–/d/d-id/1337578.

[6] A. Derhab, M. Guerroumi, A. Gumaei et al., "Blockchain and random subspace learning-based ids for sdn-enabled industrial iot security," *Sensors*, vol. 19, no. 14, p. 3119, 2019.

[7] M. Imran, M. H. Durad, F. A. Khan, and A. Derhab, "Toward an optimal solution against denial of service attacks in software defined networks," *Future Generation Computer Systems*, vol. 92, pp. 444–453, 2019.

[8] B. Du, H. Peng, S. Wang et al., "Deep irregular convolutional residual lstm for urban traffic passenger flows prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 972–985, 2020.

[9] E. Bou-Harb, M. Debbabi, and C. Assi, "Big data behavioral analytics meet graph theory: on effective botnet takedowns," *IEEE Network*, vol. 31, no. 1, pp. 18–26, 2017.

[10] E. M. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Scalable and robust unsupervised android malware fingerprinting using community-based network partitioning," *Computers & Security*, vol. 96, article 101932, 2020.

[11] M. Marjani, F. Nasaruddin, A. Gani et al., "Big IOT data analytics: architecture, opportunities, and open research challenges," *IEEE Access*, vol. 5, pp. 5247–5261, 2017.

[12] A. Aldweesh, A. Derhab, and A. Z. Emam, "Deep learning approaches for anomaly-based intrusion detection systems: a survey, taxonomy, and open issues," *Knowledge-Based Systems*, vol. 189, article 105124, 2020.

[13] M. A. Ferrag, L. Maglaras, S. Moschoyiannis, and H. Janicke, "Deep learning for cyber security intrusion detection: approaches, datasets, and comparative study," *Journal of Information Security and Applications*, vol. 50, article 102419, 2020.

[14] S. Mahdavifar and A. A. Ghorbani, "Application of deep learning to cybersecurity: A survey," *Neurocomputing*, vol. 347, pp. 149–176, 2019.

[15] Z. Ning, P. Dong, X. Wang et al., "Mobile edge computing enabled 5g health monitoring for internet of medical things: a decentralized game theoretic approach," *IEEE Journal on Selected Areas in Communications*, pp. 1–16, 2020.

[16] Z. Ning, P. Dong, X. Wang et al., "Partial computation offloading and adaptive task scheduling for 5g-enabled vehicular networks," *IEEE Transactions on Mobile Computing*, 2020.

[17] Z. Ning, K. Zhang, X. Wang et al., "Intelligent edge computing in internet of vehicles: a joint computation offloading and caching solution," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[18] Z. Ning, K. Zhang, X. Wang et al., "Joint computing and caching in 5g-envisioned internet of vehicles: a deep reinforcement learning-based traffic control system," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[19] T. Wang, G. Zhang, A. Liu, M. Z. A. Bhuiyan, and Q. Jin, "A secure iot service architecture with an efficient balance dynamics based on cloud and edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4831–4843, 2018.

[20] X. Wang, Z. Ning, and S. Guo, "Multi-agent imitation learning for pervasive edge computing: a decentralized computation offloading algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 411–425, 2020.

[21] X. Wang, Z. Ning, S. Guo, and L. Wang, "Imitation learning enabled task scheduling for online vehicular edge computing," *IEEE Transactions on Mobile Computing*, 2020.

[22] A. Derhab, M. Belaoued, M. Guerroumi, and F. A. Khan, "Two-factor mutual authentication offloading for mobile cloud computing," *IEEE Access*, vol. 8, pp. 28956–28969, 2020.

[23] A. Boulemtafes, A. Derhab, and Y. Challal, "A review of privacy-preserving techniques for deep learning," *Neurocomputing*, vol. 384, pp. 21–45, 2020.

[24] A. A. Diro and N. Chilamkurti, "Distributed attack detection scheme using deep learning approach for internet of things," *Future Generation Computer Systems*, vol. 82, pp. 761–768, 2018.

[25] M. A. Ferrag and L. Maglaras, "DeepCoin: a novel deep learning and blockchain-based energy exchange framework for smart grids," *IEEE Transactions on Engineering Management*, vol. 67, no. 4, pp. 1285–1297, 2020.

[26] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for internet of things," *IEEE Access*, vol. 5, pp. 18042–18050, 2017.

[27] Y. Otoum, D. Liu, and A. Nayak, "DL-IDS: a deep learning–based intrusion detection framework for securing IoT," *Transactions on Emerging Telecommunications Technologies*, no. - article e3803, 2019.

[28] M. Kumar, *Deep learning approach for intrusion detection system (ids) in the internet of things (iot) network using gated recurrent neural networks (gru)*, Wright State University, 2017.

[29] M. Roopak, G. Y. Tian, and J. Chambers, "Deep learning models for cyber security in IoT networks," in *2019 IEEE 9th annual computing and communication workshop and conference (CCWC)*, pp. 452–457, Las Vegas, NV, USA, 2019.

[30] M. Roopak, G. Y. Tian, and J. Chambers, "An intrusion detection system against ddos attacks in iot networks," in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 562–567, Las Vegas, NV, USA, 2020.

[31] B. Roy and H. Cheung, "A deep learning approach for intrusion detection in internet of things using bi-directional long short-term memory recurrent neural network," in *2018 28th International Telecommunication Networks and Applications Conference (ITNAC)*, pp. 1–6, Sydney, NSW, Australia, 2018.

[32] G. Thamilarasu and S. Chawla, "Towards deep-learning-driven intrusion detection for the internet of things," *Sensors*, vol. 19, no. 9, article 1977, 2019.

[33] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, https://arxiv.org/abs/1609.04747.

[34] S. Hou, A. Saas, L. Chen, and Y. Ye, "Deep4maldroid: a deep learning framework for android malware detection based on linux kernel system call graphs," in *2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW)*, pp. 104–111, Omaha, NE, USA, 2016.

[35] E. M. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Maldozer: automatic framework for android malware detection using deep learning," *Digital Investigation*, vol. 24, pp. S48–S59, 2018.

[36] T. G. Kim, B. J. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for android malware detection using various features," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773–788, 2019.

[37] S. Ni, Q. Qian, and R. Zhang, "Malware identification using visualization images and deep learning," *Computers & Security*, vol. 77, pp. 871–885, 2018.

[38] G. Sun and Q. Qian, "Deep learning and visualization for identifying malware families," *IEEE Transactions on Dependable and Secure Computing*, p. 1, 2018.

[39] Z. Wang, J. Cai, S. Cheng, and W. Li, "Droiddeeplearner: identifying android malware using deep learning," in *2016 IEEE 37th Sarnoff Symposium*, pp. 160–165, Newark, NJ, USA, 2016.

[40] S. M. Kasongo and Y. Sun, "A deep learning method with wrapper based feature extraction for wireless intrusion detection system," *Computers & Security*, vol. 92, article 101752, 2020.

[41] F. A. Khan, A. Gumaei, A. Derhab, and A. Hussain, "TSDL: a two-stage deep learning model for efficient network intrusion detection," *IEEE Access*, vol. 7, pp. 30373–30385, 2019.

[42] Z. Li, Q. Zheng, P. Shen, and L. Jiang, "Intrusion detection using temporal convolutional networks," in *International Conference on Neural Information Processing*, pp. 168–178, Sydney, NSW, Australia, 2019.

[43] W.-H. Lin, P. Wang, B.-H. Wu, M.-S. Jhou, K.-M. Chao, and C.-C. Lo, "Behaviorial-based network flow analyses for anomaly detection in sequential data using temporal convolutional networks," in *Advances in E-Business Engineering for Ubiquitous Computing. ICEBE 2019*, pp. 173–183, Springer, 2019.

[44] E. Min, J. Long, Q. Liu, J. Cui, and W. Chen, "Tr-ids: anomaly-based intrusion detection through text-convolutional neural network and random forest," *Security and Communication Networks*, vol. 2018, 9 pages, 2018.

[45] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, 2018.

[46] Q. Zhou, J. Wu, and L. Duan, "Recommendation attack detection based on deep learning," *Journal of Information Security and Applications*, vol. 52, article 102493, 2020.

[47] J. C. Bansal, H. Sharma, S. S. Jadon, and M. Clerc, "Spider monkey optimization algorithm for numerical optimization," *Memetic Computing*, vol. 6, no. 1, pp. 31–47, 2014.

[48] M. Kumar and C. Guria, "The elitist non-dominated sorting genetic algorithm with inheritance (i-NSGA- II) and its jumping gene adaptations for multi-objective optimization," *Information Sciences*, vol. 382-383, pp. 15–37, 2017.

[49] 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.

[50] S. Aldhaheri, D. Alghazzawi, L. Cheng, B. Alzahrani, and A. Al-Barakati, "Deepdca: novel network-based detection of iot attacks using artificial immune system," *Applied Sciences*, vol. 10, no. 6, p. 1909, 2020.

[51] Y. N. Soe, P. I. Santosa, and R. Hartanto, "Ddos attack detection based on simple ann with smote for iot environment," in *2019 Fourth International Conference on Informatics and Computing (ICIC)*, pp. 1–5, Semarang, Indonesia, 2019.

[52] M. Ge, F. Xiping, 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)*, pp. 256–25609, Kyoto, Japan, 2019.

[53] A. L.-H. Muna, N. Moustafa, and E. Sitnikova, "Identification of malicious activities in industrial internet of things based on deep learning models," *Journal of Information security and applications*, vol. 41, pp. 1–11, 2018.

[54] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018, https://arxiv.org/abs/1803.01271.

[55] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas, "Handling imbalanced datasets: a review," *GESTS International Transactions on Computer Science and Engineering*, vol. 30, no. 1, pp. 25–36, 2006.

[56] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[57] W. McKinney, *Pydata development team pandas: powerful python data analysis toolkit*, 2019.

[58] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, MIT press, 2016.

[59] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas, "Data preprocessing for supervised leaning," *International Journal of Computer Science*, vol. 1, no. 2, pp. 111–117, 2006.

[60] M. Al-Zewairi, S. Almajali, and A. Awajan, "Experimental evaluation of a multi-layer feed-forward artificial neural network classifier for network intrusion detection system," in *2017 International Conference on New Trends in Computing Sciences (ICTCS)*, pp. 167–172, Amman, Jordan, 2017.

[61] 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.