International Information and Engineering Technology Association
Advancing the World of Information and Engineering

International Journal of Safety and Security Engineering
Vol. 13, No. 4, August, 2023, pp. 715-722
Journal homepage: http://iieta.org/journals/ijsse

Deep Learning-Based Detection of IoT Botnet Attacks: An Exploration of Residual Networks

Manikandan Govindaraji[1*], Ramachandran Periyasamy[2], Vidyaathulasiraman[3]

[1] Department of Master of Computer Applications, Priyadarshini Engineering College Vaniyambadi, Tamil Nadu 635751, India
[2] Department of Information Technology, KG College of Arts and Science, Coimbatore, Tamil Nadu 641035, India
[3] Department of Computer Science, Government Arts and Science College for Women, Bargur, Tamil Nadu 635104, India

Corresponding Author Email: drmanikandang@priyadarshini.net.in

https://doi.org/10.18280/ijsse.130414

Received: 6 March 2023
Revised: 31 July 2023
Accepted: 11 August 2023
Available online: 28 September 2023

Keywords:
deep learning techniques, intrusion detection systems, convolution neural networks, long short-term memories, machine learning techniques, IoTs (internet of things), IoT botnets, IoT botnet detections, anomaly detections

ABSTRACT

Modern enterprises increasingly employ Internet of Things (IoT) devices across various sectors to enhance service provision, with applications spanning from healthcare to academia. However, the widespread adoption of IoT technology introduces significant security vulnerabilities. Particularly, these devices are susceptible to cyber-attacks, notably those orchestrated by botnets. The challenge of addressing this security issue is further compounded by the devices' memory and energy constraints, which limit the implementation of robust security measures. The present study introduces a Deep Learning Techniques (DLT) based approach, termed Detection of Intrusions in IoT using Residual Networks (DIIOTRNs), to preemptively identify IoT botnet attacks. These attacks typically undergo several stages prior to execution, providing an opportunity for early detection. The proposed DIIOTRNs framework integrates Convolution Neural Networks (CNNs) and Long Short-Term Memories (LSTMs) to effectively detect potential threats. The framework was subjected to empirical testing and demonstrated promising results, achieving accuracy levels exceeding 90%. Thus, the DIIOTRNs approach offers a promising solution to the pressing issue of IoT security, particularly in the context of botnet attacks. Further research is warranted to refine and optimize this framework for broad adoption across the IoT landscape.

1. INTRODUCTION

The proliferation of computer network applications has significantly influenced socio-economic development, with impact areas including international trade, healthcare, and military operations. This highlights the paramount importance of network security, which is continually threatened by both internal and external factors. Intrusion Detection Systems (IDSs) [1] have been employed since the 1980s [2] as a potential countermeasure. IDSs operate by detecting threats through the analysis of network data patterns [3]. Denial of Service attacks (DoSs), for instance, manipulate harmful traffic to obstruct or limit legitimate users' access to network resources [4]. Other disruptive elements include malware [5], utilized by attackers to compromise system stability. IDSs serve as potential solutions to mitigate the impacts of such attacks. Figure 1 illustrates an IDS model.

Data collection from systems and networks is conducted by IDSs for threat analysis [6]. Upon threat detection, corrective action is initiated and all significant network events are logged by the IDS [7]. Autonomous anomaly detection systems identify irregularities by progressively uncovering aberrant system properties [8]. Their effectiveness lies in the ability to discern abnormal behaviors within networks.

Machine Learning Techniques (MLTs) can augment IDSs by detecting attacks without human intervention. MLTs consist of an array of techniques that automatically recognize patterns and anticipate future trends [9]. Although diverse, all MLTs fundamentally operate by selecting optimal features. For example, MLTs can monitor packet sizes and distributions to discern intrusions. However, network managers often face challenges in managing intrusion reports when IDSs identify false attacks. Therefore, the enhancement of IDSs to improve detection accuracy and reduce False Alarm Rates (FARs) remains a critical area of research [10].

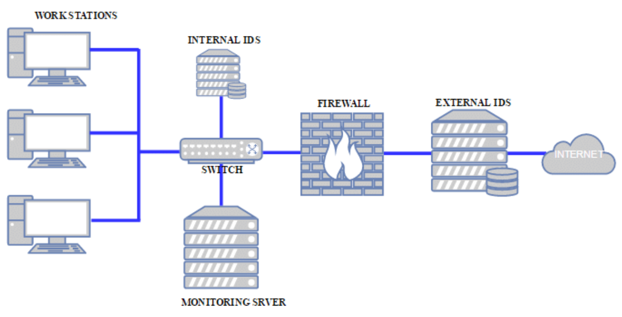


Figure 1. Mode of IDS

The problem is further exacerbated when IDSs rely on known attack signatures, leaving them incapable of identifying unknown attack types. To address this, autonomous IDSs employing MLTs as classifiers have been developed. Yet, these solutions also present drawbacks, such as limited throughput and high False Detection Rates (FDRs).

Following this introduction, botnets are discussed in Section

3. A review of related literature is presented in Section 4, followed by a description of the research methodology in Section 5. Section 6 presents the key findings from the experiments, while Section 7 evaluates the proposed scheme. Finally, Section 8 concludes the research.

1.1 Definition of the problem

The proliferation of Internet of Things (IoT) devices has been paralleled by an escalating susceptibility to cyber-attacks, attributable to the sophisticated nature of malware, most notably, IoT botnets. The constantly evolving nature of botnets renders them elusive to traditional and signature-based anomaly detection methods. While numerous Machine Learning Techniques (MLTs) have been proposed for detecting IoT botnets, a conspicuous gap in predicting potential vulnerabilities persists.

1.2 IoT botnets

IoT devices, which are anticipated to establish approximately 83 billion connections by 2024, are integral to data-dependent innovations, amassing extensive volumes of data and enabling a myriad of tasks across various sectors, including academia, residential applications, and healthcare. Despite the benefits of continuous connectivity and accessibility, IoT devices offer an opportunistic platform for hackers to execute attacks such as Distributed Denial of Services (DDoSs). IoT botnets, which refer to a network of infected computers known as bots under the control of an administrator or "botmaster" [11], pose a significant threat to the use of IoT devices. Various types of malware specifically designed for IoT devices have been deployed, with IoT botnets being the primary targets [12]. Notable examples of botnets include Bashlight, Mirai, and Torii.

Botnets operate in several stages, as illustrated in Figure 2, each stage marked by various destructive actions [13]. Initially, attackers exploit vulnerabilities in IoT devices to install bots, which are then used for malicious purposes. While awaiting further instructions and simultaneously scanning for new vulnerable devices, the bots maintain contact with the botmaster, facilitating the expansion of the botnet.

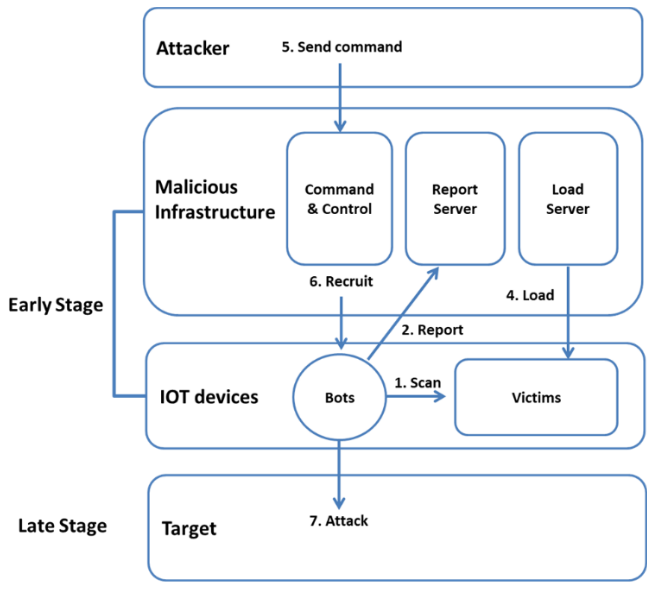


Figure 2. Stages of botnet

The focus of attack detection is generally on identifying attack behaviors that occur post-command initiation, when the attackers instruct the IoT botnets to commence the attack. Moreover, the implementation of IoT security measures contributes to increased power and memory demands. Given the vulnerability of IoT devices to botnet attacks, the development of robust defense strategies and procedures is essential. The process of creating botnets involves multiple stages, each necessitating a unique detection method. An examination of the detection strategies employed at each stage is crucial, given the diverse activities involved. There is a lack of research on early detection of IoT botnets despite the fact that early botnet stages evolve over time into more complex stages where rapid attack operations occur. Recent activities of Mirai versions have escalated [14], underscoring the importance of detecting IoT botnets. This work contributes to the existing body of knowledge on IoT botnet detection by analyzing the behavior of IoT malware using dataset examples. If implemented, the proposed methodology could significantly enhance a system's ability to detect IoT botnets.

2. REVIEW OF LITERATURE

This section delves into the taxonomies of IoT botnet detection, providing a synopsis and critique of recent research in the field. Convolutional Neural Networks (CNNs), a type of Deep Learning Technique (DLT), are designed to automatically extract and learn features from inputs [15]. Traditionally leveraged for visual information analysis, CNNs have been co-opted into cybersecurity to accurately identify malicious behaviors. For instance, the CNN model has been deployed to detect Denial-of-Service (DoS) attacks and intrusions. Long Short-Term Memory units (LSTMs) address the issue of vanishing gradients through the employment of specialized units known as "memory cells," which represent long-term memory [16]. Several studies have utilized LSTMs for various botnet detection tasks. A comprehensive literature review encompassing the stages of botnet formation and detection was undertaken [13]. The study provided an insight into IoT botnet detection. IoT botnet frameworks were established in the research conducted by Stephens et al. [17], providing a foundation for future exploration and potential solutions. Machine Learning Techniques (MLTs) and DLTs have been employed to identify botnets in various studies. MLTs were used to categorize benign and malicious behaviors, employing a combination of Decision Trees (DTs), K Nearest Neighbors (KNNs), Random Forests (RFs), and Support Vector Machines (SVMs) for processing tasks [18]. Deep Learning Techniques such as FastGRNN, LSTMs, and GRU were evaluated by Giaretta et al. [19], focusing on the identification of infected and compromised devices. Machine Learning Techniques were devised to detect IoT devices affected by botnets [20]. Using an IoT dataset embedded with botnet attacks (Bashlite and Mirai) from various IoT devices, the study proposed a botnet detection model that utilized CNNs, achieving an F1-score of 91%. Jung et al. [21] proposed a Deep Learning Technique based on CNNs, incorporating a data-processing component. Energy usage was standardized and segmented to enhance the accuracy of CNNs. The model categorized the processed data into four classes, including a botnet class. Cross-device evaluation on three common types of IoT devices yielded a cross-test accuracy of 90%. The leave-one-out examination demonstrated an accuracy greater than 90%, and the overall assessment boasted an accuracy rate of 96.5%. Furthermore, an examination of IoT botnets using MLTs such as Decision Trees (DTs), Association Rule Mining (ARM), Naive Bayes (NBs), and Artificial Neural Networks (ANNs) on the UNSW-NB15 dataset was conducted by Koroniotis et al. [22]. The study's results indicated that Decision Trees improved detection procedures by 93%, as evidenced by the accuracy and false alarm rate.

3. PROPOSED DIIOTRNS SCHEME

AIs (Artificial intelligences), MLTs, and DLTs can all be used in cybersecurity to build powerful tools that detect and stop hostile behaviours in networks. MLTs may be used to analyze, detect, and comprehend complicated patterns in data, as well as forecast future outcomes. Models learn as they proceed and use this knowledge to improve their capacity to both recognize and foresee probable appearance of forthcoming cyber attacks. The benefits of DLTs over traditional MLTs include their higher performances in a number of settings, notably while learning from massive security datasets. This work's proposed DIIOTRNs is based on CNNs and LSTMs for detecting botnets from IoTs devices. The proposed model can significantly improve IoT botnet detection abilities of systems when implemented. CNNs are discriminative layered DLTs which use one or more convolution and pooling layers as arrays to create multilayer NNs (Neural Networks) [23]. The convolution layers often share weights, and pooling layer samples to produce some sort of translational invariant feature outputs. The proposed scheme is based on CNNs and executed on IoT botnet dataset and depicted as Figure 3.

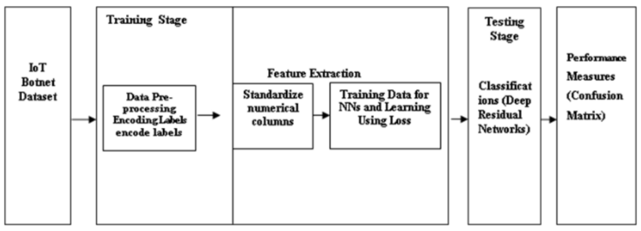
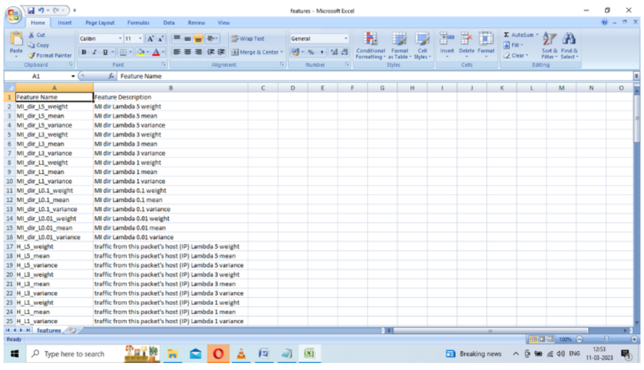

IoT Botnet Dataset
Training Stage
Data Pre-processing EncodingLabels encode labels
Feature Extraction
Standardize numerical columns
Training Data for NNs and Learning Using Loss
Testing Stage
Classifications (Deep Residual Networks)
Performance Measures (Confusion Matrix)

Figure 3. DIIOTRNs scheme

DIIOTRN is executed on the IoT Botnet dataset following the processes of data pre-processing/preparation where data is cleaned and labels are encoded. This is subsequently followed by CNNs based feature extractions where numerical columns are standardized and features are selected based error or loss rates while learning from data. The final stages are classifications of botnet data packets. The suggested schema was evaluated in terms of precision, recall and f1-measure values along with a confusion matrix. The dataset, preprocesses, feature selections and classifications of DIIOTRNs are detailed below.

3.1 Dataset

A multivariate sequential dataset including actual traffic data from nine commercial IoT devices infected with BASHLITE and Mirai was utilised in the investigation and downloaded from Kaggle.com the internet [24]. IoT related malware behave differently from others, minimal dataset sizes and qualities of dataset have a substantial influence on functions of DLTs. Figure 4 displays a sample of the dataset.

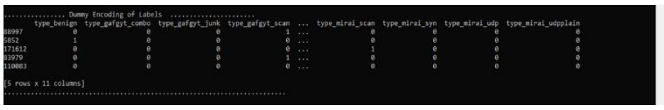


	A	B
1	Feature Name	Feature Description
2	MI_dir_L5_weight	MI dir Lambda 5 weight
3	MI_dir_L5_mean	MI dir Lambda 5 mean
4	MI_dir_L5_variance	MI dir Lambda 5 variance
5	MI_dir_L3_weight	MI dir Lambda 3 weight
6	MI_dir_L3_mean	MI dir Lambda 3 mean
7	MI_dir_L3_variance	MI dir Lambda 3 variance
8	MI_dir_L1_weight	MI dir Lambda 1 weight
9	MI_dir_L1_mean	MI dir Lambda 1 mean
10	MI_dir_L1_variance	MI dir Lambda 1 variance
11	MI_dir_L0.1_weight	MI dir Lambda 0.1 weight
12	MI_dir_L0.1_mean	MI dir Lambda 0.1 mean
13	MI_dir_L0.1_variance	MI dir Lambda 0.1 variance
14	MI_dir_L0.01_weight	MI dir Lambda 0.01 weight
15	MI_dir_L0.01_mean	MI dir Lambda 0.01 mean
16	MI_dir_L0.01_variance	MI dir Lambda 0.01 variance
17	H_L5_weight	traffic from this packet's host (IP) Lambda 5 weight
18	H_L5_mean	traffic from this packet's host (IP) Lambda 5 mean
19	H_L5_variance	traffic from this packet's host (IP) Lambda 5 variance
20	H_L3_weight	traffic from this packet's host (IP) Lambda 3 weight
21	H_L3_mean	traffic from this packet's host (IP) Lambda 3 mean
22	H_L3_variance	traffic from this packet's host (IP) Lambda 3 variance
23	H_L1_weight	traffic from this packet's host (IP) Lambda 1 weight
24	H_L1_mean	traffic from this packet's host (IP) Lambda 1 mean
25	H_L1_variance	traffic from this packet's host (IP) Lambda 1 variance

Figure 4. Data set sample

3.2 DIIOTRN's preprocessing

Label encoding is the process of converting labels into numeric representations that machines can read. MLTs can then better predict the functioning of such labels where data gets labeled while training for clarity of data. It is a significant structured dataset supervised learning pre-processing step. As an example: In some dataset, assume Height. After applying label encoding, the Height column is changed into where 0 represents tall, 1 represents medium, and 2 represents low height. Figure 5 depicts dummy encoding of labels by DIIOTRNs.



```
............... Dummy Encoding of Labels  ......................
        type_benign  type_gafgyt_combo  type_gafgyt_junk  type_gafgyt_scan  ...  type_mirai_scan  type_mirai_syn  type_mirai_udp  type_mirai_udpplain
88997             0                  0                 0                 1  ...                0               0               0                    0
5852              1                  0                 0                 0  ...                0               0               0                    0
171612            0                  0                 0                 0  ...                1               0               0                    0
83979             0                  0                 0                 1  ...                0               0               0                    0
110083            0                  0                 0                 0  ...                0               0               0                    0

[5 rows x 11 columns]
.......................................................................
```

Figure 5. Dummy label encoding

3.3 DIIOTRN's feature extractions/selections

When developing predictive models, feature selection procedures entail reducing input variable counts for improving model performances while simultaneously reducing costs in models. In statistically based feature selection approaches, relationships of input variables with target variables are evaluated, and input variables with strongest associations are selected. These approaches execute quickly and efficiently in spite of the fact that they are statistical measures that are dependent on t input and output data types. Figure 6 displays feature extractions and selections.

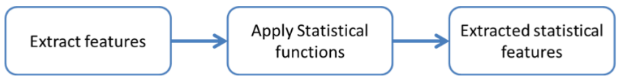


Figure 6. Feature selections and extractions

3.4 DIIOTRNs classification

CNNs are multilayer neural networks which are that are discriminative DLTs and are made up of one or more convolution and pooling layers [23]. Convolution layers often share a large number of weights with pooling layer samples to produce some sort of translational invariant features. CNNs feature fewer parameters than other connected networks with same hidden unit counts, making training easier. CNN architecture incorporates biologically inspired MLPs (multi layer Perceptrons) [25]. The term "receptive field" alludes to the small number of visual sub-regions to which these cells are sensitive. These fields are arranged to occupy the whole visual field, allowing the cell to function as a local filter throughout the entire input area. CNN's architecture includes convolution, max pooling, and fully linked layers. The convolution layer is made up of neurons placed in a rectilinear grid, as opposed to preceding layers, which were made up of neurons arranged in a rectangular grid. The rectangular grid neurons are linked together by a network of weights known as filter banks, which receive inputs from preceding rectangular units. In order to generate convolution layers, the weights for the rectangular units must remain constant for each rectangular grid of neurons. Each grid utilises a separate filter bank in systems where the convolution layer is made up of many grids. A pooling layer comes after each convolution layer, merging subsets of the rectangular block formed by the convolution layer by taking subsamples to provide an output of the block. In order to pool the neurons in the blocks, one can compute their maximum, average, or learn a linear summing algorithm. Some blocks move more than a row or column, which provide an input to nearby pooling units. As a result, the system's dimensions are reduced [26]. In the last phase of, convolution and max-pooling layers are non-linearly stacked to generate a completely connected layer. The connection that makes training a set of weights for a filter bank straightforward in the last phase of the neural network, convolution and max-pooling layers are non-linearly stacked to generate a completely connected layer which makes it simple to train a set of weights for filter banks. ResNets address the problem of performance degradations related to DNNs (deep neural networks) with their deeper explorations of networks. Utilizing the strength of voluminous data by building DNNs that try to approximate functions f(x) and map inputs, x to their corresponding labels, y. According to the Universal Approximation Theorem, every function across a domain may be closely approximated by a feedforward network with a single layer if the layer contains enough neurons or hidden units. However, in real-world applications, deeper L-layer neural networks with fewer hidden units per layer are able to approximate functions that exponentially more hidden units are needed to calculate in shallower levels. As the signal advances further into the layers, the actual strength of DLTs comes in their capacity to capture abstract characteristics. However, the well-known vanishing/exploding gradients and performance deterioration problem frequently affects deeper neural networks. The accuracy reaches a saturation point as network depths rise, after which it starts declining. Analyses on higher error rates of CNNs revealed that they were caused by vanishing/exploding gradients. Researchers at Microsoft Research proposed ResNets (Residual Networks) in 2015, a novel architecture, to overcome this issue where residual Blocks created in this design. Residual blocks are stacks of layers configured in way where layer outputs get added to subsequent layers down the stack. Non-linearity gets applied after combining outputs of equivalent layers. The bye passes are named shortcuts or skip-connections which are used in this work and depicted in Figure 7. By skipping a few intermediary levels and producing leftover blocks, skip connections link the activations of one layer to the succeeding layers. These discarded blocks are piled to form ResNets. The goal of this network is for networks to fit residual mappings rather than for layers to learn the underlying mapping. Regularization will skip any layer that reduces architectural performance if this type of skip link is included. As a result, training highly DNNs is achievable without running into problems with vanishing or growing gradients. These skip connections, like LSTMs, make use of parametric gates.

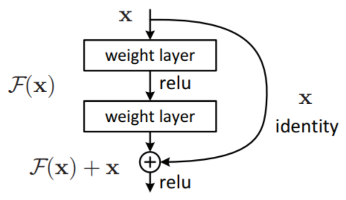

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 7. Skip connections

The gates determine how much information passes through skip connections and based on dimensions two kinds of residual blockscan exist namely Identical residual blocks and Convolution residual blockswhich are depicted in Figure 8.

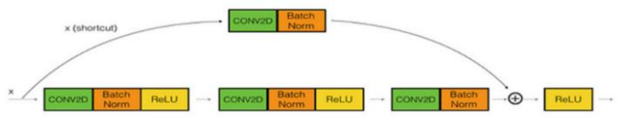
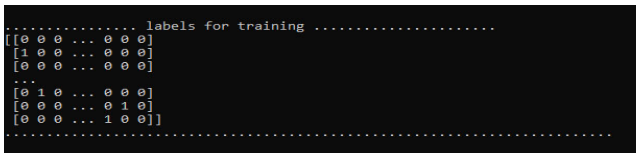


Figure 8. Residual blocks

The outputs of shortcut paths and main paths are both identical residual blocks with same dimensions. This is achieved when the input of each convolution layer in the main route is padded such that the output and input dimensions remain consistent. In skip-connections of convolution residual blocks, a convolution layer is used to enlarge the output of the shortcut path to the same dimension as the main path. To regulate the output loudness, the layer may also employ different filter sizes, such as padding, strides, and 11 filters. The fundamental purpose of the convolution layer is to apply a learnt linear function that lowers the dimension of the input. Non-linear functions are not used. However, this design did not give more precision. It makes no use of non-linear functions. However, this design did not provide more accuracy than the ResNet architecture. Use a multi-layered network design inspired by VGG-19 (50), to which the shortcut connection is later added. The design is then transformed into a residual network via these short-cut links. A residual network is created by stacking many residual blocks together. To slow down performance, the residual blocks build an identity mapping to previous network activations. Deep neural networks have a degradation concern.

4. RESULTS AND DISCUSSION

Results for each step of the planned DIIOTRNs programme are presented as shown in Figure 9. The experimental setup for the suggested model in this study was developed in Python version 3.9.0, a sophisticated language for data science with a number of helpful modules and a leading programming language for embedded systems, such as IoT devices. The dataset used was obtained from 9 IoT devices and training sets of each device recorded typical network traffic patterns. Each device's test data was made up of all the malicious data as well as the remaining third of benign data. The dataset can also be utilized for multi-class classifications as it encompasses 10 classes of assaults plus a class of "benign." Malicious data can be separated into 10 attacks undertaken by 2 botnets. Network packets separated into distinct classes are listed in Table 1.



```
................ labels for training .....................
[[0 0 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 1 0 ... 0 0 0]
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 1 0 0]]
.......................................................................
```

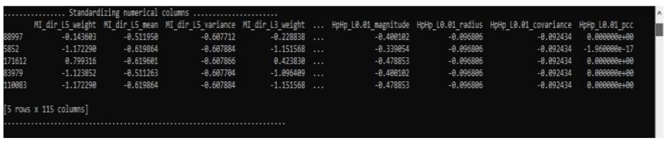
Figure 9. DIIOTRNs label encoding

Table 1. Dataset instances

Data Class	Number of Instances
benign	62154
gafgyt_combo	61380
gafgyt_junk	30898
gafgyt_scan	29297
gafgyt_tcp	104510
gafgyt_udp	104011
mirai_ack	60554
mirai_scan	96781
mirai_syn	65746
mirai_udp	156248
mirai_udpplain	56681

4.1 DIIOTRNs preprocessing

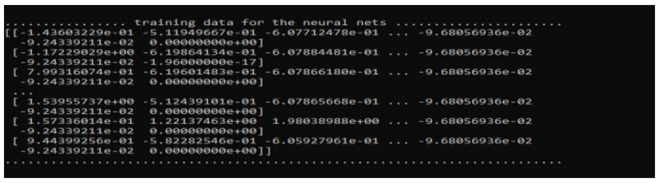
Label encoding is the primary component of the dataset preparation procedure in this work. MLTs typically involve working with datasets that include numerous labels in one or more columns. These designations may be written in words or represented by numbers. Figure 10 depicts label encoding.



```
................ Standardizing numerical columns .......................
       MI_dir_L5_weight  MI_dir_L5_mean  MI_dir_L5_variance  MI_dir_L3_weight  ...  HpHp_L0.01_magnitude  HpHp_L0.01_radius  HpHp_L0.01_covariance  HpHp_L0.01_pcc
88997         -0.143603       -0.511950           -0.607712         -0.228838  ...             -0.400102          -0.096806              -0.092434    0.000000e+00
5852          -1.172290       -0.619864           -0.607884         -1.151568  ...             -0.339054          -0.096806              -0.092434   -1.960000e-17
171612         0.799316       -0.619601           -0.607866          0.423830  ...             -0.478853          -0.096806              -0.092434    0.000000e+00
83979         -1.123852       -0.511263           -0.607704         -1.096409  ...             -0.400102          -0.096806              -0.092434    0.000000e+00
110083        -1.172290       -0.619864           -0.607884         -1.151568  ...             -0.478853          -0.096806              -0.092434    0.000000e+00

[5 rows x 115 columns]
.......................................................................
```

Figure 10. DIIOTRN's normalizations



```
................ training data for the neural nets .......................
[[-1.43603229e-01 -5.11949667e-01 -6.07712478e-01 ... -9.68056936e-02
  -9.24339211e-02  0.00000000e+00]
 [-1.17229029e+00 -6.19864134e-01 -6.07884481e-01 ... -9.68056936e-02
  -9.24339211e-02 -1.96000000e-17]
 [ 7.99316074e-01 -6.19601483e-01 -6.07866180e-01 ... -9.68056936e-02
  -9.24339211e-02  0.00000000e+00]
 ...
 [ 1.53955737e+00 -5.12439101e-01 -6.07865668e-01 ... -9.68056936e-02
  -9.24339211e-02  0.00000000e+00]
 [ 1.57336014e-01  1.22137463e+00  1.98038988e+00 ... -9.68056936e-02
  -9.24339211e-02  0.00000000e+00]
 [ 9.44399256e-01 -5.82282546e-01 -6.05927961e-01 ... -9.68056936e-02
  -9.24339211e-02  0.00000000e+00]]
.......................................................................
```

Figure 11. Training data for neural nets

4.2 DIIOTRN's feature extractions/selections

The features are divided into four types, which summarize traffics between hosts and protocols in communications.

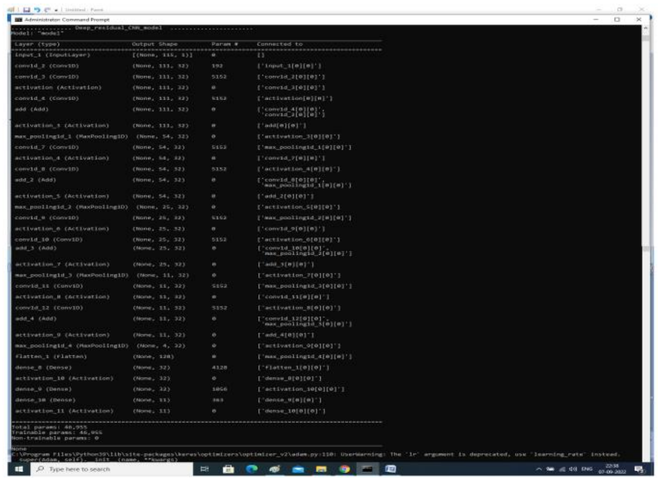
- Traffic generated by same IP addresses (Type 1): Count, mean, and variation of host MAC and IP-Packets (3).
- Traffic generated by same IP and MAC addresses (Type 2): Mean, variance, magnitude, radius, covariance, and correlation of channel packets (7).
- Traffics between same sources and destinations (Type 3): Packet Jitter in a Network Packet count, mean, and variation of packet jitter in a channel (3).
- TCP/UDP communications between same sources and destinations (Type 4): Mean, variance, magnitude, radius, covariance, and correlation of socket packets (7).

Before training, the numerical columns were shuffled, standardized, and normalized, and the dataset's records is shown in Figure 11.

4.3 DIIOTRN's classification

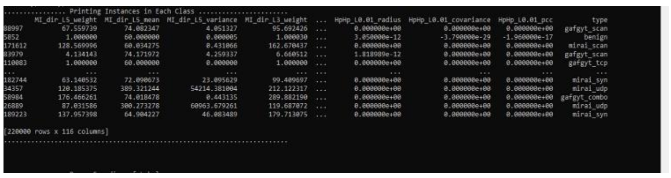
To estimate DLT performance for predictive modeling problems, divide the dataset into training and test datasets. This study employed the train test split technique to divide the dataset into training and test data in a 70:20 ratio with 10% for validations. Figure 12 and Figure 13 depict data trained for neural nets.

The suggested hybrid model has several layers, including inputs, CNNs, LSTMs, flat/ dense/output layers. The first layer's CNNs got input from 128, 64 neurons, whereas the second layer's LSTMs received input from 32, 16 neurons. The dense layers had 128, 64 neurons as they result in high levels of accuracy, these two layers were integrated in the model. In the flatten layer, the vector is flattened into a one-dimensional vector that may be utilized in the dense layer.



```
................ Deep_residual_CNN_model  ......................
Model: "model"
____________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
====================================================================================
 input_1 (InputLayer)           [(None, 115, 1)]     0           []
 conv1d_2 (Conv1D)              (None, 111, 32)      192         ['input_1[0][0]']
 conv1d_3 (Conv1D)              (None, 111, 32)      5152        ['conv1d_2[0][0]']
 activation (Activation)        (None, 111, 32)      0           ['conv1d_3[0][0]']
 conv1d_4 (Conv1D)              (None, 111, 32)      5152        ['activation[0][0]']
 add (Add)                      (None, 111, 32)      0           ['conv1d_4[0][0]',
                                                                  'conv1d_2[0][0]']
 activation_3 (Activation)      (None, 111, 32)      0           ['add[0][0]']
 max_pooling1d_1 (MaxPooling1D)  (None, 54, 32)      0           ['activation_3[0][0]']
 conv1d_7 (Conv1D)              (None, 54, 32)       5152        ['max_pooling1d_1[0][0]']
 activation_4 (Activation)      (None, 54, 32)       0           ['conv1d_7[0][0]']
 conv1d_8 (Conv1D)              (None, 54, 32)       5152        ['activation_4[0][0]']
 add_2 (Add)                    (None, 54, 32)       0           ['conv1d_8[0][0]',
                                                                  'max_pooling1d_1[0][0]']
 activation_5 (Activation)      (None, 54, 32)       0           ['add_2[0][0]']
 max_pooling1d_2 (MaxPooling1D)  (None, 25, 32)      0           ['activation_5[0][0]']
 conv1d_9 (Conv1D)              (None, 25, 32)       5152        ['max_pooling1d_2[0][0]']
 activation_6 (Activation)      (None, 25, 32)       0           ['conv1d_9[0][0]']
 conv1d_10 (Conv1D)             (None, 25, 32)       5152        ['activation_6[0][0]']
 add_3 (Add)                    (None, 25, 32)       0           ['conv1d_10[0][0]',
                                                                  'max_pooling1d_2[0][0]']
 activation_7 (Activation)      (None, 25, 32)       0           ['add_3[0][0]']
 max_pooling1d_3 (MaxPooling1D)  (None, 11, 32)      0           ['activation_7[0][0]']
 conv1d_11 (Conv1D)             (None, 11, 32)       5152        ['max_pooling1d_3[0][0]']
 activation_8 (Activation)      (None, 11, 32)       0           ['conv1d_11[0][0]']
 conv1d_12 (Conv1D)             (None, 11, 32)       5152        ['activation_8[0][0]']
 add_4 (Add)                    (None, 11, 32)       0           ['conv1d_12[0][0]',
                                                                  'max_pooling1d_3[0][0]']
 activation_9 (Activation)      (None, 11, 32)       0           ['add_4[0][0]']
 max_pooling1d_4 (MaxPooling1D)  (None, 4, 32)       0           ['activation_9[0][0]']
 flatten_1 (Flatten)            (None, 128)          0           ['max_pooling1d_4[0][0]']
 dense_8 (Dense)                (None, 32)           4128        ['flatten_1[0][0]']
 activation_10 (Activation)     (None, 32)           0           ['dense_8[0][0]']
 dense_9 (Dense)                (None, 32)           1056        ['activation_10[0][0]']
 dense_10 (Dense)               (None, 11)           363         ['dense_9[0][0]']
 activation_11 (Activation)     (None, 11)           0           ['dense_10[0][0]']
====================================================================================
Total params: 46,955
Trainable params: 46,955
Non-trainable params: 0
____________________________________________________________________________________
None
C:\Program Files\Python39\lib\site-packages\keras\optimizers\optimizer_v2\adam.py:110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

Figure 12. DIIOTRN's Deep residual_CNN model



```
................ Printing Instances in Each Class .......................
       MI_dir_L5_weight  MI_dir_L5_mean  MI_dir_L5_variance  MI_dir_L3_weight  ...  HpHp_L0.01_radius  HpHp_L0.01_covariance  HpHp_L0.01_pcc          type
88997         67.559739       74.082347            4.051327         95.692426  ...       0.000000e+00           0.000000e+00    0.000000e+00   gafgyt_scan
5852           1.000000       60.000000            0.000005          1.000030  ...       3.050000e-12          -3.790000e-29   -1.960000e-17        benign
171612       128.569996       60.034275            0.431066        162.670437  ...       0.000000e+00           0.000000e+00    0.000000e+00    mirai_scan
83979          4.134143       74.171972            4.259337          6.660512  ...       1.818989e-12           0.000000e+00    0.000000e+00   gafgyt_scan
110083         1.000000       60.000000            0.000000          1.000000  ...       0.000000e+00           0.000000e+00    0.000000e+00    gafgyt_tcp
...                 ...             ...                 ...               ...  ...                ...                    ...             ...           ...
182744        63.140532       72.098673           23.095629         99.409697  ...       0.000000e+00           0.000000e+00    0.000000e+00     mirai_syn
34357        120.185375      389.321244        54214.381004        212.122317  ...       0.000000e+00           0.000000e+00    0.000000e+00     mirai_udp
58984        176.466261       74.018478            0.443135        289.882190  ...       0.000000e+00           0.000000e+00    0.000000e+00  gafgyt_combo
26889         87.031586      300.273278        60963.679261        119.687072  ...       0.000000e+00           0.000000e+00    0.000000e+00     mirai_udp
189223       137.957398       64.904227           46.083489        179.713075  ...       0.000000e+00           0.000000e+00    0.000000e+00     mirai_syn

[220000 rows x 116 columns]
.......................................................................
```

Figure 13. Trained class instances

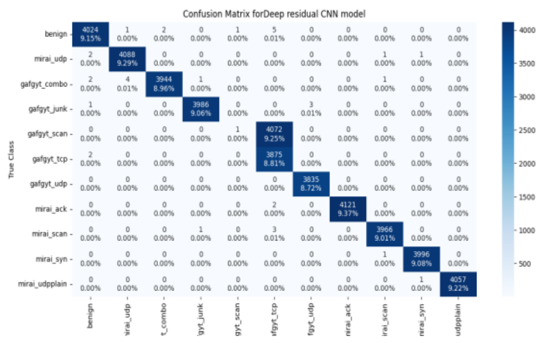


Figure 14. Confusion matrix of DIIOTRN's deep residual CNN model

To avoid model's over fits, the model includes a dropout layer with a rate of 0.2, which is done by randomly removing certain neurons from the final layer. The output is created in the dense layer using the ReLU activation algorithm. The researcher built the model using a categorical cross-entropy loss function for multiclass classification and a binary categorical cross-entropy loss function for binary classification. Figure 14 displays a snapshot of Trained class instances.

5. EVALUATION METRICS

A confusion matrix based on four assessment indicators was used to evaluate the model: The letter TP (True Positive) denotes that the proposed model correctly predicts the positive class; the letter TN (True Negative) denotes that the proposed model correctly predicts the negative class; the letter FP (False Positive) denotes that the proposed model incorrectly predicts the positive class; and the letter FN denotes (False Negative) that the proposed model incorrectly predicts the negative class. The confusion matrix of DIIOTRN's Deep residual CNN model is shown in Figure 15. Based on these measures, the precisions, recalls, and F1-scores shown below were evaluated:

Precision: Precision measures the number of positive class predictions, which are truly from the positive class. It is formulated as given,

$$Precision = \frac{TP}{FP + TP} \tag{1}$$

Recall: Recall provides the measure of the number of positive class predictions obtained out of all the positive examples present in the dataset. It is expressed as below,

$$Recall = \frac{TP}{FN + TP} \tag{2}$$

F1-Score: F1-Score yields a single score, which helps balancing both the issues of precision and recall in one number. It is expressed as below,

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3}$$

Accuracy: Accuracy refers to the ratio of examples, which were rightly classified. In exact terms, it is the ratio between the sum of the number of true positives and true negatives, andthe number of examples present in the dataset. It is formulated as given,

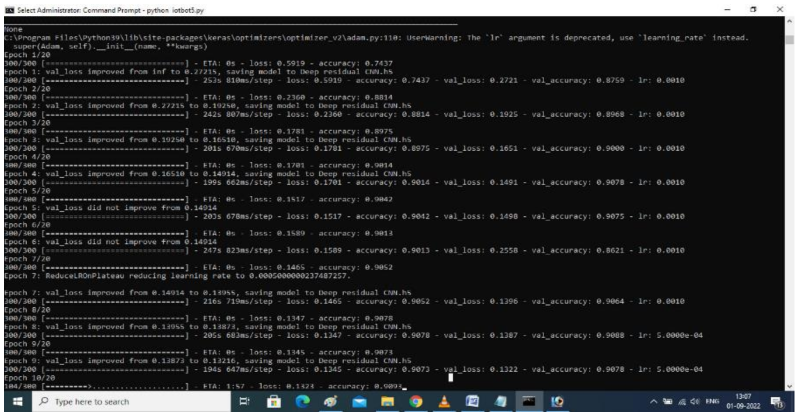
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

CNNs may significantly reduce the amount of parameters, which improves model learning efficiency. Furthermore, LSTMs have their own memory and can produce rather accurate classifications. As a result, the Cross CNN LSTMs design employs CNN layers to extract features from input data and combines them with LSTMs to facilitate prediction. We can see from the preceding data that the suggested Cross CNN LSTMs model has a high detection rate. Figure 16 depicts the results of CNNs.

Table 2 shows the experimental findings of DIIOTRNs running 20 and 50 epochs. The findings and measures for precisions, recalls, and F1-scores for binary and multiclass classifications are shown in the table below. Traffic was classified in binary as malicious and legitimate.

Table 2. DIIOTRNs performance metric values for 20 and 50 epochs runs

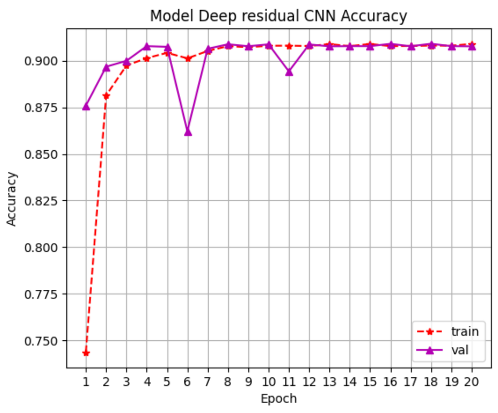
Class	precision		recall		f1-score		support	
	20 Epochs	50 Epochs	20 Epochs	50 Epochs	20 Epochs	50 Epochs	20 Epochs	50 Epochs
benign	1.00	1.00	1.00	1.00	1.00	1.00	4033	3967
mirai_udp	1.00	1.00	1.00	1.00	1.00	1.00	4092	3963
gafgyt_combo	1.00	1.00	1.00	1.00	1.00	1.00	3952	4087
gafgyt_junk	1.00	1.00	1.00	1.00	1.00	1.00	3990	4041
gafgyt_scan	0.50	0.00	0.00	1.00	1.00	1.00	4073	4013
gafgyt_tcp	0.49	1.00	0.65	0.50	1.00	0.67	3877	4064
gafgyt_udp	1.00	1.00	1.00	1.00	1.00	1.00	3835	3981
mirai_ack	1.00	1.00	1.00	1.00	1.00	1.00	4123	3980
mirai_scan	1.00	1.00	1.00	1.00	1.00	1.00	3970	4038
mirai_syn	1.00	1.00	1.00	1.00	1.00	1.00	3997	3952
mirai_udpplain	1.00	1.00	1.00	1.00	1.00	1.00	4058	3914


Select Administrator: Command Prompt - python iotbot5.py

```
None
C:\Program Files\Python39\lib\site-packages\keras\optimizers\optimizer_v2\adam.py:110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
Epoch 1/20
300/300 [==============================] - ETA: 0s - loss: 0.5919 - accuracy: 0.7437
Epoch 1: val_loss improved from inf to 0.27215, saving model to Deep residual CNN.h5
300/300 [==============================] - 253s 810ms/step - loss: 0.5919 - accuracy: 0.7437 - val_loss: 0.2721 - val_accuracy: 0.8759 - lr: 0.0010
Epoch 2/20
300/300 [==============================] - ETA: 0s - loss: 0.2360 - accuracy: 0.8814
Epoch 2: val_loss improved from 0.27215 to 0.19250, saving model to Deep residual CNN.h5
300/300 [==============================] - 242s 807ms/step - loss: 0.2360 - accuracy: 0.8814 - val_loss: 0.1925 - val_accuracy: 0.8968 - lr: 0.0010
Epoch 3/20
300/300 [==============================] - ETA: 0s - loss: 0.1781 - accuracy: 0.8975
Epoch 3: val_loss improved from 0.19250 to 0.16510, saving model to Deep residual CNN.h5
300/300 [==============================] - 201s 670ms/step - loss: 0.1781 - accuracy: 0.8975 - val_loss: 0.1651 - val_accuracy: 0.9000 - lr: 0.0010
Epoch 4/20
300/300 [==============================] - ETA: 0s - loss: 0.1701 - accuracy: 0.9014
Epoch 4: val_loss improved from 0.16510 to 0.14914, saving model to Deep residual CNN.h5
300/300 [==============================] - 199s 662ms/step - loss: 0.1701 - accuracy: 0.9014 - val_loss: 0.1491 - val_accuracy: 0.9078 - lr: 0.0010
Epoch 5/20
300/300 [==============================] - ETA: 0s - loss: 0.1517 - accuracy: 0.9042
Epoch 5: val_loss did not improve from 0.14914
300/300 [==============================] - 203s 678ms/step - loss: 0.1517 - accuracy: 0.9042 - val_loss: 0.1498 - val_accuracy: 0.9075 - lr: 0.0010
Epoch 6/20
300/300 [==============================] - ETA: 0s - loss: 0.1589 - accuracy: 0.9013
Epoch 6: val_loss did not improve from 0.14914
300/300 [==============================] - 247s 823ms/step - loss: 0.1589 - accuracy: 0.9013 - val_loss: 0.2558 - val_accuracy: 0.8621 - lr: 0.0010
Epoch 7/20
300/300 [==============================] - ETA: 0s - loss: 0.1465 - accuracy: 0.9052
Epoch 7: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

Epoch 7: val_loss improved from 0.14914 to 0.13955, saving model to Deep residual CNN.h5
300/300 [==============================] - 216s 719ms/step - loss: 0.1465 - accuracy: 0.9052 - val_loss: 0.1396 - val_accuracy: 0.9064 - lr: 0.0010
Epoch 8/20
300/300 [==============================] - ETA: 0s - loss: 0.1347 - accuracy: 0.9078
Epoch 8: val_loss improved from 0.13955 to 0.13873, saving model to Deep residual CNN.h5
300/300 [==============================] - 205s 683ms/step - loss: 0.1347 - accuracy: 0.9078 - val_loss: 0.1387 - val_accuracy: 0.9088 - lr: 5.0000e-04
Epoch 9/20
300/300 [==============================] - ETA: 0s - loss: 0.1345 - accuracy: 0.9073
Epoch 9: val_loss improved from 0.13873 to 0.13216, saving model to Deep residual CNN.h5
300/300 [==============================] - 194s 647ms/step - loss: 0.1345 - accuracy: 0.9073 - val_loss: 0.1322 - val_accuracy: 0.9078 - lr: 5.0000e-04
Epoch 10/20
104/300 [=========>....................] - ETA: 1:57 - loss: 0.1323 - accuracy: 0.9093
```

Figure 15. Output of residual CNN model



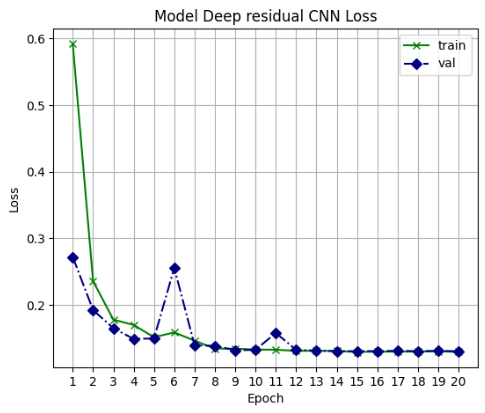


Figure 16. DIIOTRNs performances in classifications

The accuracy for a 20 epochs run was 0.91 (support 44000). The macro and weighted average for precision and recall was 0.91 while for f1-score it was 0.88 with the same support. The total batches were 1375 with a step loss of loss: 0.1303. The test accuracy was 0.906659. For a 50 epoch run accuracy was increased by 1%. Accuracy was 0.91 (support 44000) while macro/weighted averages for precision recall and f1-score were 0.88, 0.91 and 0.88 respectively. The loss and test accuracy values did not differ much and were 0.1309 and 0.9084 respectively. Hence the suggested scheme achieved an accuracy score above 90% in general. The binary classification for the two classes, legitimate and malicious, had 91% accuracy. Figure 16 demonstrates the new proposed model's accuracy and learning from losses in detecting propagations of botnet in IoT networks.

6. CONCLUSION

IoTs are being utilised to improve services in a range of commercial sectors. These applications cover a wide range of topics, including health and education. The devices facilitate and accelerate cybersecurity attacks, particularly when botnets are involved. Thankfully, botnets go through many processes before launching attacks, which might be used to detect these attacks early on. This paper suggested and implemented DIIOTRNs, a DLT-based technique for detecting IoT botnet attacks. The proposed model in this paper includes a 0.2 dropout layer to prevent model overfitting, which is achieved by randomly eliminating neurons from the final layer. The ReLU activation technique is used to generate the dense layer output. The researcher created models for multiclass and binary classifications using a categorical cross-entropy loss function and binary categorical cross-entropy, respectively. Furthermore, the model was trained for 50 epochs before being stopped after 20 epochs because the loss did not improve. An Adam optimiser and the Reduce LR On Plateau function were used to alter the learning rate. Technical empirical experiments were performed on a prototype supplied in this paper to investigate the behaviour of IoT malwares. This prototype offers a clear knowledge of the early stages of creating an IoT botnet. While it makes more sense to focus on the early stages, when the botnet is established and spreads over time, most previous research tended to focus on the late stage, which occurs quickly, providing substantial challenges in recognizing IoT botnets and blocking DDoS attacks. Furthermore, the scientists created multiclass classification algorithms for spotting IoT botnets using a real IoT dataset and a fusion model based on DLTs known as Cross CNN LSTMs. Numerous tests were carried out, and a comparison was conducted between the proposed methodology and previous works that used both standard MLTs and specific DLTs. The primary goal of this study was to investigate IoT botnet malware and understand its behaviour by monitoring and collecting network data. In order to investigate IoT botnet proliferation, this experiment focused on scanning, brute-forcing, downloading, and installing malware binaries on IoT devices. A novel paradigm for recognizing IoT botnets was proposed, and implemented using a dataset and assessed with several performance criteria of F1-score, accuracy, and recall. The results of the studies indicate that the recommended model is precise and reaches accuracy levels of above 90%. However, limitations arise from dataset representativeness, model robustness against adversarial attacks, and real-time implementation challenges. Additionally, scalability concerns and resource constraints in resource-limited IoT devices need to be addressed for practical deployment. Future improvement in DIIOTRNs can focus on diversifying datasets and incorporating adversarial defense mechanisms for better generalization and resilience against attacks, while optimizing for real-time deployment and scalability in resource-limited IoT environments.

REFERENCES

[1] Chand, N., Mishra, P., Krishna, C.R., Pilli, E.S., Govil, M.C. (2016). A comparative analysis of SVM and its stacking with other classification algorithm for intrusion detection. 2016 International Conference on Advances in Computing, Communication, & Automation (ICACCA) (Spring), pp. 1-6. https://doi.org/10.1109/ICACCA.2016.7578859

[2] SANS Institute. (2016). The history and evolution of intrusion detection. Available: https://www.sans.org/reading-room/whitepapers/detection/history-evolution-intrusion-detection-344, accessed on Feb. 20, 2016.

[3] Jonnalagadda, S.K., I., R.P.R. (2013). A literature survey and comprehensive study of intrusion detection. International Journal of Computer Applications, 81(16): 40-47. https://doi.org/10.5120/14210-2458

[4] Mitchell, R., Chen, I.R. (2014). A survey of intrusion detection techniques for cyber-physical systems. ACM

Computing Surveys, 46(4): 1-29. https://doi.org/10.1145/2542049

[5] Internet of Things: How much are we exposed to cyber threats? Infosec Resources. https://resources.infosecinstitute.com/topics/iot-security/internet-things-much-exposed-cyber-threats/, accessed on Dec. 10, 2015.

[6] Hampshire. IoT connections to reach 83 billion by 2024, driven by maturing industrial use cases. https://www.juniperresearch.com/press/iot-connections-to-reach-83-bn-by-2024, accessed on Apr. 7, 2022.

[7] What it is Network intrusion detection system? http://www.combofix.org/what-it-is-network-intrusion-detection-system.php, accessed on Dec. 10, 2015.

[8] Denning, D.E. (1987). An intrusion-detection model. IEEE Transactions on Software Engineering, SE-13(2): 222-232. https://doi.org/10.1109/TSE.1987.232894

[9] Alpaydın, E. (2010). Introduction To Machine learning. https://www.lri.fr/~xlzhang/KAUST/CS229_slides/chapter18_RL.pdf, accessed on Jan. 20, 2015.

[10] Masduki, B.W., Ramli, K., Saputra, F.A., Sugiarto, D. (2015). Study on implementation of machine learning methods combination for improving attacks detection accuracy on Intrusion Detection System (IDS). 2015 International Conference on Quality in Research (QiR), Lombok, Indonesia, pp. 56-64. https://doi.org/10.1109/QiR.2015.7374895

[11] Alzahrani, H., Abulkhair, M., Alkayal, E. (2020). A multi-class neural network model for rapid detection of IoT botnet attacks. International Journal of Advanced Computer Science and Applications (IJACSA), 11(7): 688-696. https://doi.org/10.14569/IJACSA.2020.0110783

[12] TrendMicro. Into the Battlefield: A Security Guide to IoT Botnets. 2019. https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/into-the-battlefield-a-security-guide-to-iot-botnets, accessed on March 5, 2021.

[13] Wazzan, M., Algazzawi, D., Bamasaq, O., Albeshri, A., Cheng, L. (2021). Internet of Things botnet detection approaches: Analysis and recommendations for future research. Applied Sciences, 11: 5713. https://doi.org/10.3390/app11125713

[14] CSDE. International Botnet and Iot Security Guide 2020. https://securingdigitaleconomy.org/wp-content/uploads/2019/11/CSDE_Botnet-Report_2020_FINAL.pdf, accessed on April 7, 2022.

[15] Li, Y.M., Xu, Y.Y., Liu, Z., Hou, H.X., Zheng, Y.S., Xin, Y.; Zhao, Y.F., Cui, L.Z. (2019). Robust detection for network intrusion of industrial IoT based on multi-CNN fusion. Measurement, 154(2): 107450. https://doi.org/10.1016/j.measurement.2019.107450

[16] Abuhamad, M., Abuhmed, T., Mohaisen, D., Nyang, D.H. (2020). AUToSen: Deep-learning-based implicit continuous authentication using smartphone sensors. IEEE Internet of Things Journal, 7(6): 5008-5020. https://doi.org/10.1109/JIOT.2020.2975779

[17] Stephens, B., Shaghaghi, A., Doss, R., Kanhere, S.S. (2021). Detecting internet of things bots: A comparative study. IEEE Access, 9: 160391-160401. https://doi.org/10.1109/ACCESS.2021.3130714

[18] Guerra-Manzanares, A., Medina-Galindo, J., Bahsi, H., Nõmm, S. (2020). MedBIoT: Generation of an IoT botnet dataset in a medium- sized IoT network. In ICISSP; ResearchGate: Berlin, Germany, pp. 207-218. https://doi.org/10.5220/0009187802070218

[19] Giaretta, L., Lekssays, A., Carminati, B., Ferrari, E., Girdzijauskas, Š. (2021). LiMNet: Early-Stage Detection of IoT Botnets with Lightweight Memory Networks. In: Bertino, E., Shulman, H., Waidner, M. (eds) Computer Security – ESORICS 2021. ESORICS 2021. Lecture Notes in Computer Science(), vol 12972. Springer, Cham. https://doi.org/10.1007/978-3-030-88418-5_29

[20] Kim, J., Shim, M., Hong, S., Shin, Y., Choi, E. (2020). Intelligent detection of IoT botnets using machine learning and deep learning. Applied Sciences, 10(19): 7009. https://doi.org/10.3390/app10197009

[21] Jung, W., Zhao, H.Y., Sun, M.L., Zhou, G. (2020). IoT botnet detection via power consumption modeling. Smart Health, 15: 100103. https://doi.org/10.1016/j.smhl.2019.100103

[22] Koroniotis, N., Moustafa, N., Sitnikova, E., Slay, J. (2017). Towards developing network forensic mechanism for botnet activities in the IoT based on machine learning techniques. In Proceedings of the International Conference on Mobile Networks and Management, Melbourne, Australia, pp. 30-44. https://doi.org/10.48550/arXiv.1711.02825

[23] LeCun, Y., Bengio, Y., Hinton, G. (2015). Deep learning. Nature, 521: 436-444. https://doi.org/10.1038/nature14539

[24] Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Breitenbacher, D., Shabtai, A., Elovici, Y. (2018). N-BaIoT: Network-based detection of IoT botnet attacks using deep autoencoders. IEEE Pervasive Computing, 17(3): 12-22. https://doi.org/10.1109/MPRV.2018.03367731

[25] Dalto, M. (2014). Deep neural networks for time series prediction with applications in ultra-short-term wind forecasting. http://www.fer.unizg.hr/_download/repository/KDI-Djalto.pdf, accessed on Feb 19, 2015.

[26] LeCun, Y. (2012). Learning Invariant Feature Hierarchies. In: Fusiello, A., Murino, V., Cucchiara, R. (eds) Computer Vision – ECCV 2012. Workshops and Demonstrations. ECCV 2012. Lecture Notes in Computer Science, vol 7583. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-33863-2_51