Article

An efficient deep learning-based detection and classification system for cyber-attacks in IoT communication networks

Qasem Abu Al-Haija * and Saleh Zein-Sabatto

Department of Electrical and Computer Engineering, Tennessee State University, Nashville, TN 37209.
* Correspondence: Qabualha@Tnstate.edu

Abstract: With the rapid expansion of intelligent resource-constrained devices and high-speed communication technologies, Internet of Things (IoT) has earned a wide recognition as the primary standard for low-power lossy networks (LLNs). Nevertheless, IoT infrastructures are vulnerable to cyber-attacks due to the constraints in computation, storage, and communication capacity of the endpoint devices. From one side, the majority of newly developed cyber-attacks are formed by slightly mutating formerly established cyber-attacks to produce a new attack tending to be treated as a normal traffic through the IoT network. From the other side, the influence of coupling the deep learning techniques with cybersecurity field has become a recent inclination of many security applications due to their impressive performance. In this paper, we provide a comprehensive development of a new intelligent and autonomous deep learning-based detection and classification system for cyber-attacks in IoT communication networks leveraging the power of convolutional neural networks, abbreviated as (IoT-IDCS-CNN). The proposed IoT-IDCS-CNN makes use of the high-performance computing employing the robust CUDA based Nvidia GPUs and the parallel processing employing the high-speed I9-Cores based Intel CPUs. In particular, the proposed system is composed of three subsystems: Feature Engineering subsystem, Feature Learning subsystem and Traffic classification subsystem. All subsystems are developed, verified, integrated, and validated in this research. To evaluate the developed system, we employed the NSL-KDD dataset which includes all the key attacks in the IoT computing. The simulation results demonstrated more than 99.3% and 98.2% of cyber-attacks' classification accuracy for the binary-class classifier (normal vs anomaly) and the multi-class classifier (five categories) respectively. The proposed system was validated using K-fold cross-validation method and was evaluated using the confusion matrix parameters (i.e., TN, TP, FN, FP) along with other classification performance metrics including precision, recall, F1-score, and false alarm rate. The test and evaluation results of the IoT-IDCS-CNN system outperformed many recent machine-learning based IDCS systems in the same area of study.

Keywords: Deep Learning, Convolutional Neural Network, IoT Networks, Cyber-attack Detection, Classification.

1. Introduction

The Internet of Things (IoT) is comprised of a collection of heterogeneous resource-constrained objects interconnected via different network architectures such as wireless sensor networks (WSN) [1]. These objects or "things" are usually composed of sensors, actuators, and processors with the ability to communicate with each other to achieve common goals/applications through unique identifiers with respect to the Internet Protocol (IP) [2, 3]. Current IoT applications include smart buildings; telecommunications; medical and pharmaceutical; aerospace and aviation; environmental phenomenon monitoring; agriculture; industrial and manufacturing processes etc. The basic IoT layered architecture is shown in Figure 1. It has three layers: the perception layer (consist of edge-devices that interact with the environment to identify certain physical factors or other smart objects in the environment), the network layer (consists of a number of networking-devices that discover and connect devices over the IoT network to transmit and receive the sensed data), and the application layer (consists of various IoT applications/services that are responsible for data processing and storage). Indeed, most cyber-attacks target the application and network layers of the IoT system.

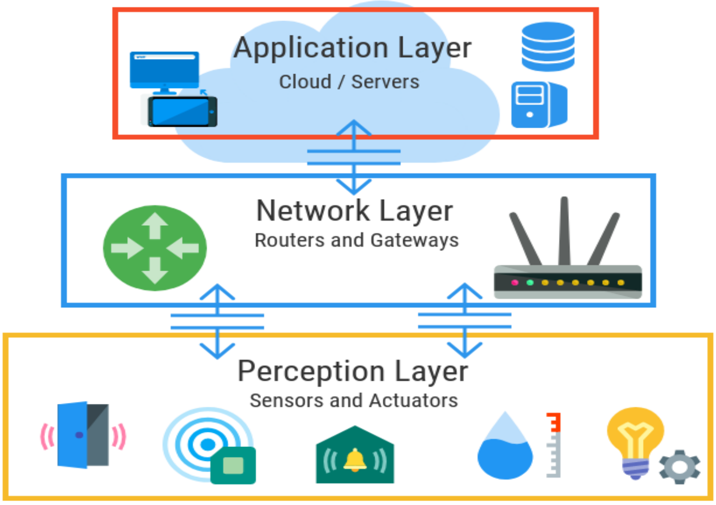


Figure 1. IoT Layered Architecture Considering the 3-layer Scheme of IoT [4].

IoT is a promising profound technology with tremendous expansion and effect. IoT infrastructures are vulnerable to cyber-attacks in that within the network, simple endpoint devices (e.g. thermostat, home appliance, etc.) are more constrained in computation, storage, and network capacity compared with the more complex endpoint devices (e.g., smartphones, laptops, etc.) that may reside within the IoT infrastructure [5, 6]. In fact, the privacy, authentication, key management, trust management and the cyber-attacks identification are among the significant challenges of the Internet of Things (IoT) and cloud based IoT [7, 8]. A number of studies were directed to address the security issues and challenges of IoT and cloud computing using block chain techniques [9, 10], light-weight authentication process [11,12], and the secure data sharing and searching of cloud based IoT [13, 14]. Once the IoT infrastructure is breached, hackers have the ability to distribute the IoT data to unauthorized parties and can manipulate the accuracy and consistency of IoT data over its entire life cycle [15]. Therefore, such cyber-attacks need to be addressed for safe IoT utilization. Consequently, vast efforts to handle the security issues in the IoT model have been made in the recent years. Many of the new cybersecurity technologies were developed by coupling the fields of machine learning with cybersecurity. It should be noted that, the majority of IoT new attacks are slight deviations (i.e. mutations) of earlier known cyberattacks [16]. Such slight mutations of these IoT attacks have been demonstrated to be difficult to identify/classify using traditional machine learning techniques. Promising state-of-art research has been conducted for cybersecurity using deep neural networks [17-22]. Table 1 summarizes research of conventional and traditional machine learning approaches to solve cybersecurity issues.

In this paper, a new intelligent system that can detect mutations of common IoT cyberattacks using non-traditional machine learning techniques exploiting the power of Nvidia-Quad GPUs is proposed. The proposed system employs the convolutional neural network (CNN) along its associated machine learning algorithms to classify the NSL-KDD dataset records (we denote our system using the acronym IoT-IDCS-CNN). The NSL-KDD dataset stores non-redundant records of all the key attacks of IoT computing with different levels of difficulties. Specifically, the main contributions of this paper can be summarized as follows:

- We provide a comprehensive efficient detection/classification model that can classify the IoT traffic records of NSL-KDD dataset into two (Binary-Classifier) or five (Multi-Classifier) classes. Also, we present detailed preprocessing operations for the collected dataset records prior to its use with deep learning algorithms.
- We provide an illustrated description of our system modules and the machine learning algorithms. Furthermore, we demonstrate a comprehensive view of the computation process of our IoT-IDCS-CNN.
- We provide an inclusive development, validation environment and configurations along with an extensive simulation results to gain insight into the proposed model and the solution approach. This includes simulation results related to the classification accuracy, classification time and classification error rate for the system validation of both detection (Binary-Classifier) and classification (Multi-Classifier).
- We provide a comprehensive performance analysis to gain more insight about the system efficiency such as the confusion matrix to analyze the attacks' detection True/False Positives and the True/False Negatives and other evaluation metrics including Precision, Recall, F-Score Metric and, False Alarm Rate.
- We benchmark study of our findings with other related state-of-art works employing the same dataset as well as the comparison with other State-of-Art machine learning based intrusion detection systems (ML-IDS) employing different dataset.

Table 1. Summary of related research for machine learning based IoT security.

Research	Method	Description
K. Taher et. al. 2019 [16]	Artificial Neural Network (ANN) with Support Vector Machine (SVM) Classifier	3-classes, with 2 hidden layers and used only 35-features
X. Gao et. al. 2019 [17]	Deep Neural Network (DNN) with ensemble voting	5-classes, 3-methods: Decision Tree, Random Forest, K-Nearest
S. Sapre, et. al. 2019 [18]	Different ML-IDS techniques	5-classes, with 2 hidden layers and Naïve Bayes Classifier
S. Jan et.al 2019 [23]	ML-IDS based SVM System	Only binary classification, used only 2 or 3 simple features
Roopak et. al. 2019 [24]	Deep Neural Network (DNN)	Small representative sample, does not reflect a realistic accuracy in actual IoT environments
Ioannou et. al 2019 [25]	ML-IDS based SVM System	Only binary classification, used anonymous sensor topology
Brun et. al, 2018 [26]	Deep Neural Network (DNN)	System validation was poorly accomplished on a testbed comprising of only three devices and naive attacks were used to validate the system using a real-time date with 50,000 samples
Thing et. al 2017 [27]	Deep Auto-Encoder (DAE)	But not realistic, very small dataset (no DDos, no Probe), 3HL (256/128/64), need significant time for FE.
Shukla et. al 2017 [28]	Neural Network Hybrid Learning (K_Means + Decision Trees)	Only binary classification, small scale simulated network (16 nodes) with different topologies
Hodo et. al 2016 [29]	Multi-Perceptron Layer (MLP) Neural Network	But not realistic, small dataset, binary classes
Kolias et. al 2016 [30]	Different ML-IDS Techniques	Very time-consuming manual feature selection, 4-classes
Y. Li et. al. 2015 [31]	Hybrid NN (Autoencoder + Deep Belief NN)	Redundant dataset needs to be up to date to reflect more rationale results.

The rest of this paper is organized as follows: Section 2 introduces and justifies the dataset of IoT cyberattacks employed by our system. Section 3 provides details of the proposed system architecture, development, and detailed design steps. Section 4 presents the simulation environment for system implementation, testing and validation. Section 5 discusses the details about experimental evaluation, comparison, and discussion. Finally, Section 6 concludes the findings of the research.

2. Dataset of Cyberattacks

Data collection involves the gathering of information on variables of interest (VOS) within a dataset in a documented organized manner that allows one to answer the defined research enquiries, examine the stated hypotheses, and assess the output consequences. In this research, the variables of interest are concerned with the intrusions/attacks data records in IoT computing environments. Two global datasets of IoT attacks can be investigated including KDD'99 dataset and NSL-KDD dataset. Indeed, KDD'99 has been developed by DARPA intrusion detection evaluation program to build a network IDS able of differentiating amongst "bad" and "good" connections [32]. This dataset includes a standard list of data to be inspected, which contains a broad range of cyber-attacks modeled in a military communication platform. However, one of the most important issues of this dataset is the enormous number of redundant data-samples in the training and testing datasets. Such redundancy affects the accuracy of classifier which will have a biased towards more frequent records [32].

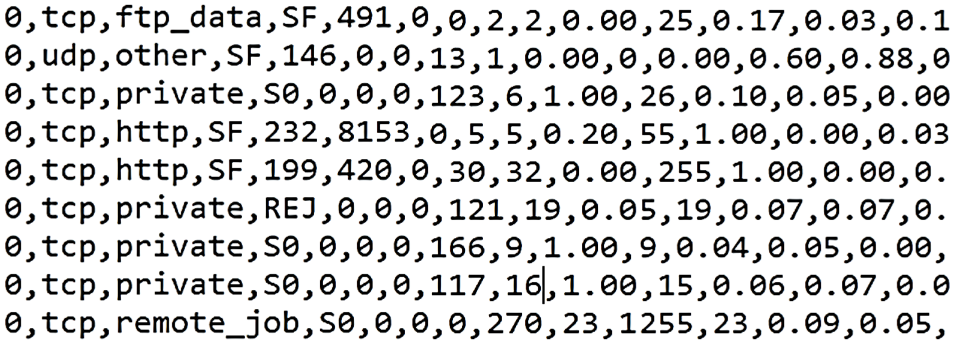
Lately, KDD'99 has been re-investigated and updated to include more up-to-date and non-redundant attack records with different levels of difficulties through the newer version called NSL-KDD [33]. NSL-KDD [34] is a reduced version of the original KDD'99 dataset [35] and consists of the same features as KDD'99. However, NSL-KDD includes more up-to-date and non-redundant attack records with different levels of difficulties. Figure 2 shows sample records of original KDD-NSL training dataset in CSV format but read by notepad in TXT format (prior to any processing technique). In this research, the NSL-KDD dataset is employed for many reasons including:

(a) It can be efficiently imported, read, preprocessed, encoded, and programed to produce two- or multi- class classification for IoT Cyber-attacks.

(b) It covers all key attacks of IoT computing including: Denial-of-Service (DoS) [36], Probe (side channel) [37], Root to Local (R2L) [38], User to Root (U2R) [38].

(c) It is obtainable as TXT/CSV file type consisting of a reasonable number of non-redundant records in the training and test sets. This improves the classification process by avoiding the bias towards more frequent records.

(d) It correlates to high-level IoT traffic structures and cyberattacks as well as it can be customized, expanded, and regenerated [34].



```
0,tcp,ftp_data,SF,491,0,0,2,2,0.00,25,0.17,0.03,0.1
0,udp,other,SF,146,0,0,13,1,0.00,0,0.00,0.60,0.88,0
0,tcp,private,S0,0,0,0,123,6,1.00,26,0.10,0.05,0.00
0,tcp,http,SF,232,8153,0,5,5,0.20,55,1.00,0.00,0.03
0,tcp,http,SF,199,420,0,30,32,0.00,255,1.00,0.00,0.
0,tcp,private,REJ,0,0,0,121,19,0.05,19,0.07,0.07,0.
0,tcp,private,S0,0,0,0,166,9,1.00,9,0.04,0.05,0.00,
0,tcp,private,S0,0,0,0,117,16,1.00,15,0.06,0.07,0.0
0,tcp,remote_job,S0,0,0,0,270,23,1255,23,0.09,0.05,
```

Figure 2. Sample records of KDD-NSL training dataset.

NSL-KDD dataset has been thoroughly developed with high-level diverse interpretations of the training data covering normal and abnormal IoT network traffic data. The normal data samples represent the legitimate data packets processed by the IoT network. The abnormal data samples represent mutated data packets (i.e., attacks) achieved by slight mutations in the previously developed attacks such as the small changes in the network packet header configurations. The original dataset is available in two classification forms: two-class traffic dataset with binary labels and multi-class traffic data set including attack-type labels and difficulty level. In both cases, it comprises 148,517 samples each with 43 attributes such as duration, protocol, service, and others [39]. The statistics of traffic distribution of NSL-KDD dataset is summarized in Table 2.

Table 2. Statistics of traffic distribution of NSL-KDD dataset [35].

	Two-Classes Dataset		Multi-Classes Dataset				
	Normal	Attack	Normal	DoS	Probe	R2L	U2R
Training	67343	58630	67343	45927	11656	995	52
Testing	9711	12833	9711	7458	2754	2421	200
Total	77,054	71,463	77,054	53,385	14,410	3,416	252

3. System Modeling

In this research, the proposed system is partitioned into distinct subsystems each of which is implemented with several modules. Specifically, the system is composed of three subsystems including: Feature Engineering (FE), Feature Learning (FL), and Detection and Classification (DC), as illustrated in Figure 3.

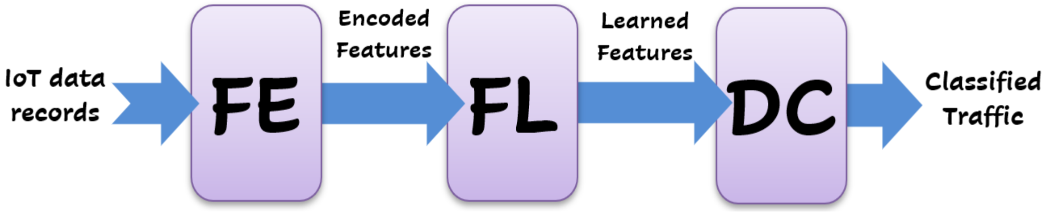
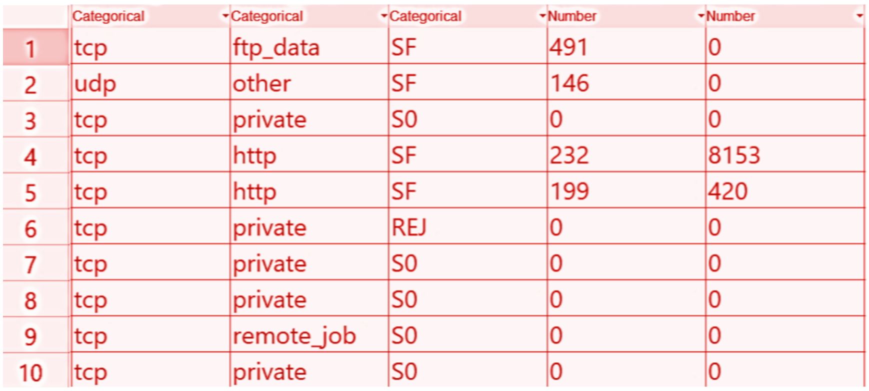


Figure 3. The three main subsystems composing the proposed system.

3.1. Implementation of Feature Engineering (FE) Subsystem

This subsystem is responsible for conversion of raw IoT traffic data records of NSL-KDD dataset into a matrix of labeled features that can be fed and trained by the neural network's part of the FL subsystem. The implementation stages of this subsystem include:

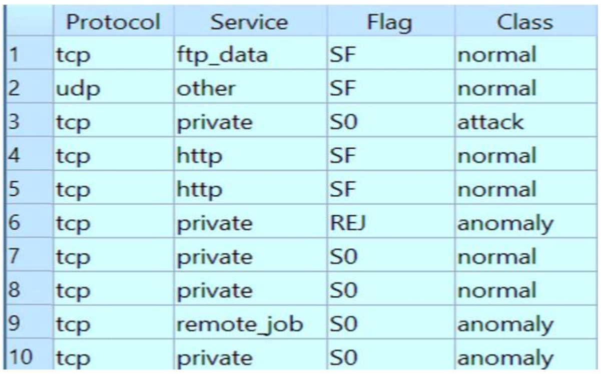
Importing NSL-KDD dataset: In this stage, the collected dataset has been imported/read using MATLAB in a tabulated format instead of raw data in the original dataset text files. All data columns are assigned virtual names based on the nature of data in the cells. The imported dataset includes 43 different features/columns. Figure 4 shows a sample of importing the NSL-KDD dataset using table datatype. The illustrated sample shows only the first ten records along with five features. All data columns were assigned virtual names based on the nature of data in the cells.



	Categorical	Categorical	Categorical	Number	Number
1	tcp	ftp_data	SF	491	0
2	udp	other	SF	146	0
3	tcp	private	S0	0	0
4	tcp	http	SF	232	8153
5	tcp	http	SF	199	420
6	tcp	private	REJ	0	0
7	tcp	private	S0	0	0
8	tcp	private	S0	0	0
9	tcp	remote_job	S0	0	0
10	tcp	private	S0	0	0

Figure 4. Importing NSL-KDD dataset: Samples from training dataset.

Renaming Categorical Features: Four of imported 43 features are categorical features that need to be renamed prior the data encoding and sample labeling processes. These features are the target protocol, the required service, the service flag, and the record category (e.g. normal or attack). Therefore, the four categorical columns have been renamed accordingly in this stage. Figure 5 illustrates the four categorical features (columns) that have been renamed accordingly for the binary-classes data records (the other columns are omitted for better readability). Also, note that the dataset encompasses multi-class data records for different traffic categories.



	Protocol	Service	Flag	Class
1	tcp	ftp_data	SF	normal
2	udp	other	SF	normal
3	tcp	private	S0	attack
4	tcp	http	SF	normal
5	tcp	http	SF	normal
6	tcp	private	REJ	anomaly
7	tcp	private	S0	normal
8	tcp	private	S0	normal
9	tcp	remote_job	S0	anomaly
10	tcp	private	S0	anomaly

Figure 5. Importing NSL-KDD dataset: Samples from training dataset.

One Hot Encoding of Categorical Features: This module is responsible for conversion of the categorical data records into numerical data records in order to be employed by the neural network. Therefore, three categorical features undergo through One Hot Encoding process (1-N encoding) [40]. These features are the protocol column, the service column, and the flag column. The class feature/column is left for samples labeling process.

- For protocol feature, three different types of protocols are revealed from the dataset including: {TCP, UDP and ICMP}. The one hot encoding for this feature will replace the categorical data of 'Protocol column' with the three numerical features as shown in Table 3.

Table 3. Scheme for Replacement of Categorical Data of Protocols

Protocol		TCP_Protocol	UDP_Protocol	ICMP_Protocol
TCP		1	0	0
UDP	Equivalent One-Hot Encoding	0	1	0
ICMP		0	0	1

- For service feature, 69 different services are revealed from the dataset such as: {'AOL', 'AUTH', 'BGP', 'COURIER', 'CSNET_NS', …, 'UUCP_PATH', 'VMNET', 'WHOIS', 'X11', 'Z39_50'}. The one hot encoding for this feature will replace the categorical data of 'Service column' with the 69 numerical features as shown in Table 4.

Table 4. Scheme for Replacement of Categorical Data of Services

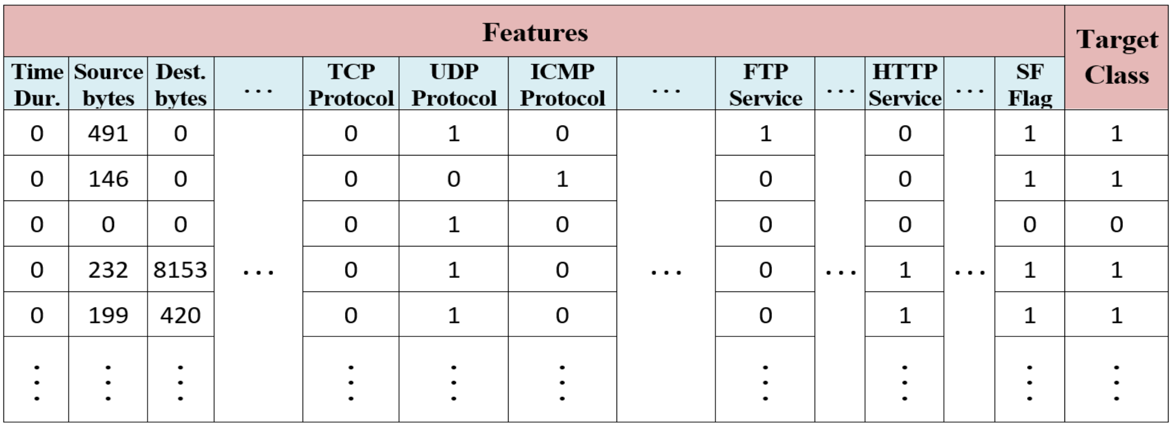
Service		AOL _ Service	AUTH_ Service	BGP_ Service	. . .	Z39_50_ Service
'AOL'		1	0	0	. . .	0
'AUTH'		0	1	0	. . .	0
'BGP'	Equivalent One-Hot Encoding	0	0	1	. . .	0
.		.	.	.		.
.		.	.	.		.
.		.	.	.	⋱	.
'Z39_50'		0	0	0	. . .	1

- For flags feature, 11 different flags are revealed from the dataset including: { 'OTH', 'REJ', 'RSTO', 'RSTOS0', 'RSTR', 'S0', 'S1', 'S2', 'S3', 'SF', 'SH' }. The one hot encoding for this feature will replace the categorical data of 'Flag column' with the 11 numerical features as shown in Table 5.

Table 5. Scheme for Replacement of Categorical Data of Flags

Flag		OTH_Flag	REJ_Flag	RSTO_Flag	. . .	SF_Flag
'OTH'		1	0	0	. . .	0
'REJ'	➡	0	1	0	. . .	0
'RSTO'		0	0	1	. . .	0
⋮	Equivalen t One-Hot Encoding	⋮	⋮	⋮	⋱	⋮
'SH'		0	0	0	. . .	1

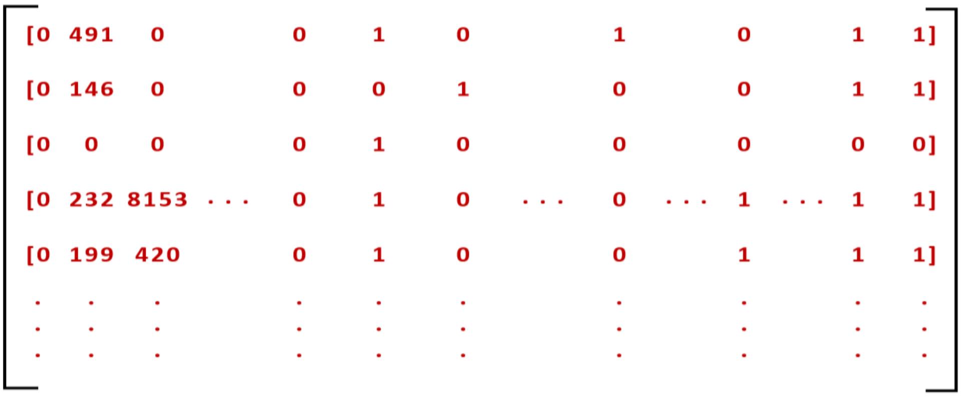
Labeling the Target Feature: This stage concerns with the samples labeling using numerical (integer) labels for the target classes. Therefore, the categorical 'Class Column' will be converted to numerical classes according to the classification technique. In our system implementation, we are considering two forms of traffic classifications: Binary classification (1: Normal vs. 2: Attack) and Multi classification (1: Normal, 2: DoS, 3:Probe, 4: R2L, 5: U2R). After this stage, all data records are available into numerical format (i.e. no categorical data exist anymore). As a result of 1-N encoding and numerical labeling, we converted the dataset into 123 features and one data label. The results of this stage, i.e. encoded form of the dataset table of 2-class records, is provided in Figure 6.



Features													Target Class
Time Dur.	Source bytes	Dest. bytes	. . .	TCP Protocol	UDP Protocol	ICMP Protocol	. . .	FTP Service	. . .	HTTP Service	. . .	SF Flag	
0	491	0	. . .	0	1	0	. . .	1	. . .	0	. . .	1	1
0	146	0		0	0	1		0		0		1	1
0	0	0		0	1	0		0		0		0	0
0	232	8153		0	1	0		0		1		1	1
0	199	420		0	1	0		0		1		1	1
⋮	⋮	⋮		⋮	⋮	⋮		⋮		⋮		⋮	⋮

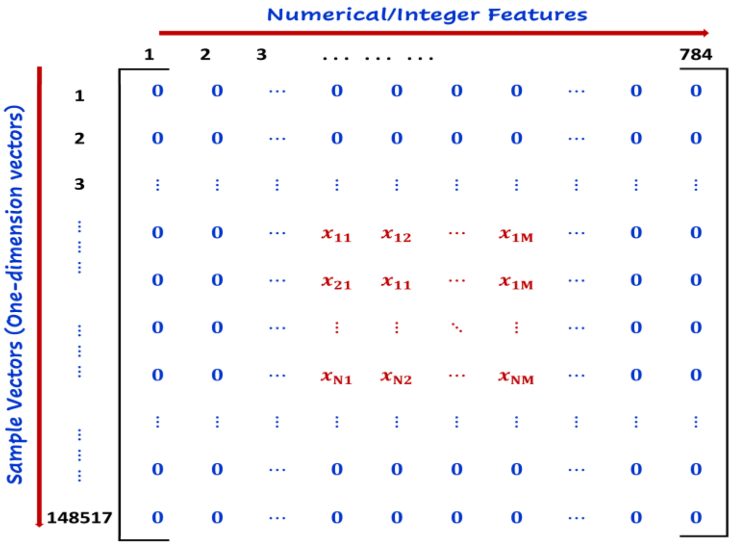
Figure 6. Encoded dataset with labeling: Sample from training set.

Converting Tables to Double Matrix: At the end of dataset importing, encoding, and labeling processes, the dataset samples and targets should be provided to the neural network inputs of FL subsystem as a matrix of all input numerical samples. Therefore, encoded dataset tables have been converted to double matrix (148517 x 124). For instance, the following double matrix illustrates the first five rows of dataset matrix.



```
[0  491    0         0    1    0         1        0       1   1]
[0  146    0         0    0    1         0        0       1   1]
[0    0    0         0    1    0         0        0       0   0]
[0  232  8153  . . . 0    1    0  . . .  0  . . . 1 . . . 1   1]
[0  199  420         0    1    0         0        1       1   1]
 .    .    .         .    .    .         .        .       .   .
 .    .    .         .    .    .         .        .       .   .
 .    .    .         .    .    .         .        .       .   .
```

Matrix Resizing with Padding Operation: This module is responsible to adjust the size of the dataset matrix to accommodate the input size for the FL subsystem. This was performed by resizing the matrix of the engineered dataset form 148517 x 124 to the new size of 148517 x 784, since the input size of every individual sample processed at FL subsystem is 28 x 28 (= 784). Thereafter, the new empty records of this matrix were padded with zero-padding technique [41]. To avoid any feature biasing in the samples of the dataset, the padded records were distributed equally around the data samples. Figure 7 illustrates an example of resizing with zero-padding operation used in this research. The new matrix size is composed of 148517 sample attack each with 784 features.


Numerical/Integer Features
Sample Vectors (One-dimension vectors)

$$\begin{array}{c|cccccccccc|}
 & 1 & 2 & 3 & \cdots & \cdots & \cdots & & & & 784 \\
1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\
2 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\
3 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & 0 & 0 & \cdots & x_{11} & x_{12} & \cdots & x_{1M} & \cdots & 0 & 0 \\
 & 0 & 0 & \cdots & x_{21} & x_{11} & \cdots & x_{1M} & \cdots & 0 & 0 \\
\vdots & 0 & 0 & \cdots & \vdots & \vdots & \ddots & \vdots & \cdots & 0 & 0 \\
 & 0 & 0 & \cdots & x_{N1} & x_{N2} & \cdots & x_{NM} & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\
148517 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0
\end{array}$$

Figure 7. Encoded dataset with labeling: Sample from training set.

Matrix Normalization with Min-Max Norm: Data normalization is performed to get all the data points to be in the same range (scale) with equal significance for each of them. Otherwise, one of the great value features might completely dominate the others in the dataset. Thus, this module is responsible to normalize all integer numbers of the dataset matrix into a range between 0~1 using Min-Max Normalization (MX-Norm) [42]. MX-Norm is well-known method to normalize data as it is commonly used in machine learning applications. In this method, we scan all the values in every feature, and then, the minimum value is converted into a 0 and the maximum value is converted into a 1, while the other values are converted (normalized) into a fraction value from 0 to 1. The Min-Max normalization (X_i^{norm}) for data record (X_i) at the (i^{th}) position of matrix (X) is defined as follows:

$$X_i^{norm} = [X_i \ - \ min(X)]/[max(X) \ - \ mi\,n(X)] \qquad (1)$$

Also, Figure 8 illustrates an example of integer data features normalized using min-max normalization (0 ~1). It can be clearly seen the effect of normalization as it ensures all features to be in the same scale.

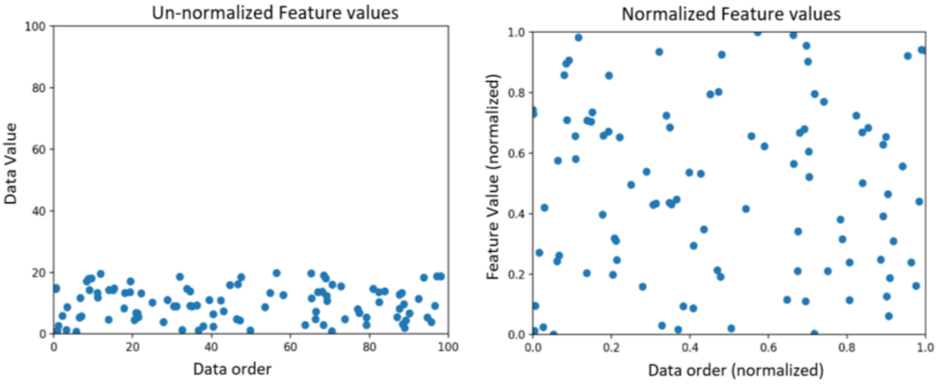


Figure 8. Illustration of Min-Max Normalization Impact over data with different scales.

Reshaping the Double Matrix: This module is responsible to create the attack samples for the $CovNet$ by reshaping the one-dimensional vectors of attack records into two-dimensional square matrices to accommodate the input size for the developed $CovNet$ network. Accordingly, every one-dimensional vector sample (1 x 784) will be reshaped into two-dimensional matrix (28 x 28) using a raw-by-raw reshaping fashion. This operation should generate a square matrix for each data sample as illustrated in Figure 9.

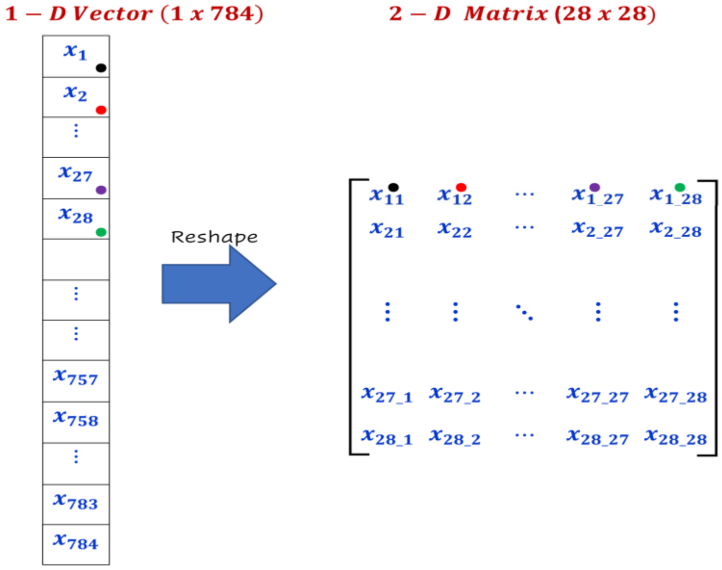

1 − D Vector (1 x 784)
2 − D Matrix (28 x 28)
Reshape
x_1
x_2
x_{27}
x_{28}
x_{757}
x_{758}
x_{783}
x_{784}
x_{11} x_{12} ⋯ x_{1_27} x_{1_28}
x_{21} x_{22} ⋯ x_{2_27} x_{2_28}
x_{27_1} x_{27_2} ⋯ x_{27_27} x_{27_28}
x_{28_1} x_{28_2} ⋯ x_{28_27} x_{28_28}

Figure 9. Illustration of Re-Shaping Operation of Dataset Samples: 1-D vector is reshaped into 2-D matric.

3.2 Implementation of Feature Learning (FL) Subsystem

So far, the FE subsystem has been developed and the next step is to process the encoded input features using FL subsystem-based CNN. The deep learning network will to be trained with minimum classification error and thus maximum accuracy. Generally, CNN involves various layers including convolution, activation, pooling, flatten and others. Convolutional layers are the core component of CNN network and they are hierarchically assembled to generate a number of feature-maps which enable CNNs to learn complex features being a vital operation to recognize patterns in the classification and detection tasks. Therefore, the developed FL subsystem is responsible for an appropriate CNN that can accept the encoded features from FE subsystem at the input layer and train on them with multiple hidden layers as well as update the training parameters before classifying the IoT traffic dataset as normal or anomaly (mutated). The implementation stages of this subsystem include:

Feature Mapping with 2D- Convolution Operations Layer: This module is responsible to generate new matrices called feature maps that emphasizes the unique features of the original matrix [43]. These feature-maps are produced by convolving (multiply and accumulate) the original matrix ($n_{in}\ x\ n_{in}$) using a number (N) of ($k\ x\ k$) convolution filters with padding size (p) and stride size of (s) which yields the feature maps ($n_{out}\ x\ n_{out}$). The size of the resultant feature maps can be evaluated as follows:

$$n_{out} = (n_{in} + 2p - k)/s + 1 \qquad (2)$$

In this research, we have applied 20 convolution filters ($9x9$) over the $28\ x\ 28$ input samples with (p=0, and $s = 1$) which resulted in 20 feature map each (20x 20). Figure 10 illustrates our convolutional layer, where the input is 28×28 matrix and a filter of size 9×9, this defines a space of 20×20 neurons in the first hidden layer. This is the case because we can only move the window 19 neurons to the right and 19 neurons to the bottom before hitting the right (or bottom) border of the input matrix. Note that the filter moves forward 1 position away, both horizontally and vertically when a new row starts. Also note that, Convolution layer goes through a backpropagation process to determine the most accurate values of its trainable parameters (weights: k x k x N = 9 x 9 x 20).

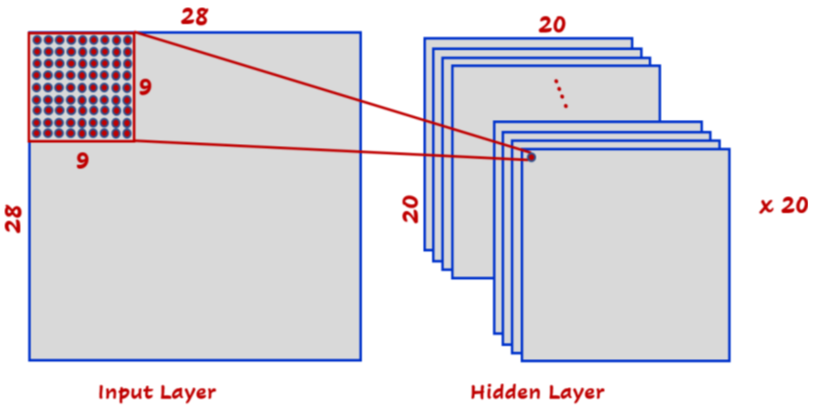


Figure 10. Implementation of convolution layer of our CNN.

Feature Activation with ReLU Function: This module is responsible to activate all units of the feature maps with non-linear rectification function namely known as ReLU. ReLU function is MAX (X,0) that sets all negative values in the matrix X to zero while all other values are kept constant. The reason of using ReLU is that training a deep network with ReLU tended to converge much more quickly and reliably than training a deep network with other non-linear activation functions such sigmoid or tanh activation functions [44]. Figure 11 illustrates the rectification layer of the convolved maps.

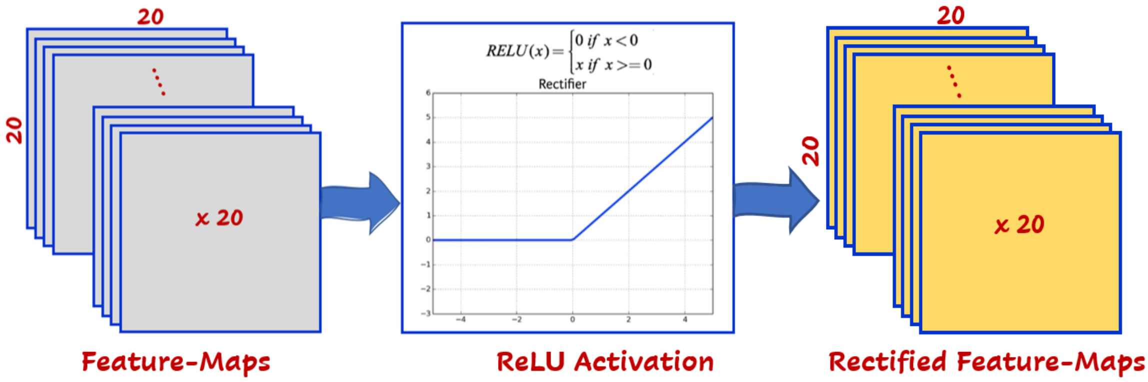


Figure 11. Implementation of ReLU activation Layer of our CNN.

Down-Sampling with Pooling Operations Layer: This module is responsible to generate new matrices called pooled feature maps that reduces the spatial size of the rectified feature maps and thus reduces the number of parameters and computation complexity in the network [43]. This can be done by combining the neighboring points of a particular region of the matrix representation into a single value that represent the selected region. The adjacent points are typically selected from a fixed size-square matrix (determined according to the application). Among these points of the applied matrix, one value is nominated as the maximum or mean of the selected points. In this research, we have used the mean pooling technique to develop the pooling layer since it combines the contribution of neighboring points instead of only selecting the maximum point. To produce the pooled feature-maps $(L_{out}\ x\ L_{out})$, the pooling filter $(f\ x\ f)$ is independently applied over the rectified feature-maps $(L_{in}\ x\ L_{in})$ with stride (s) as follows:

$$L_{out} = (L_{in} - f)/s + 1 \qquad (3)$$

In this research, we have applied 20 pooling operation $(2\ x\ 2)$ over the $20\ x\ 20$ rectified feature-maps with $(s = 2)$ which resulted in 20 feature map each $(10\ x\ 10)$. Figure 12 illustrates our pooling layer, where the input from previous layer is 20×20 x 20 and the mean pooling filter of size 2×2. Note that the stride value is 2 which means that the filter moves forward 2 positions away, both horizontally and vertically when a new row starts. Thus, we end up with pooled maps of 10×10 x 20.

Peer-reviewed version available at *Electronics* **2020**, *9*, 2152; doi:10.3390/electronics9122152

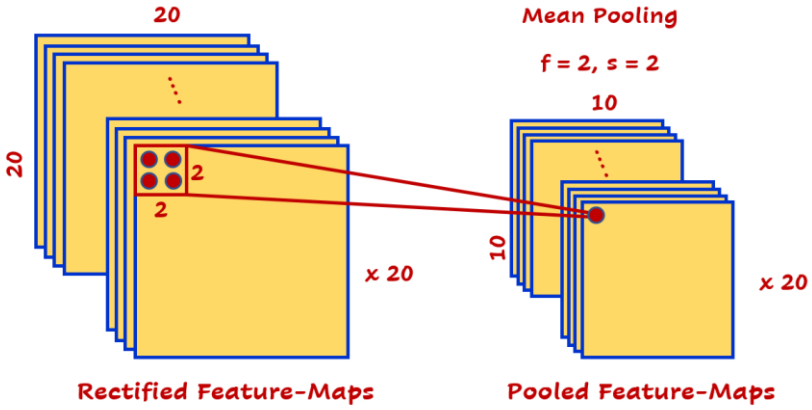


Figure 12. Implementation of pooling layer of our CNN.

3.3 Implementation of Detection and Classification (DC) Subsystem

DC subsystem is responsible for providing traffic classification for the input traffic data into binary-class classification (2-Classes: normal vs. anomaly) or multi-class classification (5-Classes: Normal, DoS, Probe, R2L, U2R). This subsystem is composed of three consecutive stages as follows:

Flattening Layer of Pooled Feature Maps: This module is responsible to linearize the output dimension of the convolutional/pooling layers network to create a single long feature vector [43]. This can be achieved by converting the 2D data of N- Pooled feature-maps into a 1-D array (or vector) to be inputted to the next layer, which is connected to the final classification model, called a dense or fully connected layer. Since flatten layer collapses the spatial dimensions of the input into the channel dimension (array), this means that if the input to the flatten layer is (N) feature maps each with a dimension of $(F_{in}\ x\ F_{in})$ then the flattened output (F_{out}) can be obtained by linear multiplication of the input dimensions by the number of maps, that's it:

$$F_{out} = N\ x\ F_{in}\ x\ F_{in} \tag{4}$$

In this research, since we have 20 pooled feature maps $(N = 20)$, each with dimension of 10 x 10 $(F_{in} = 10)$, then, our flatten layer comprise of 2000 nodes. Figure 13 illustrates the flattening layer development of our CNN.

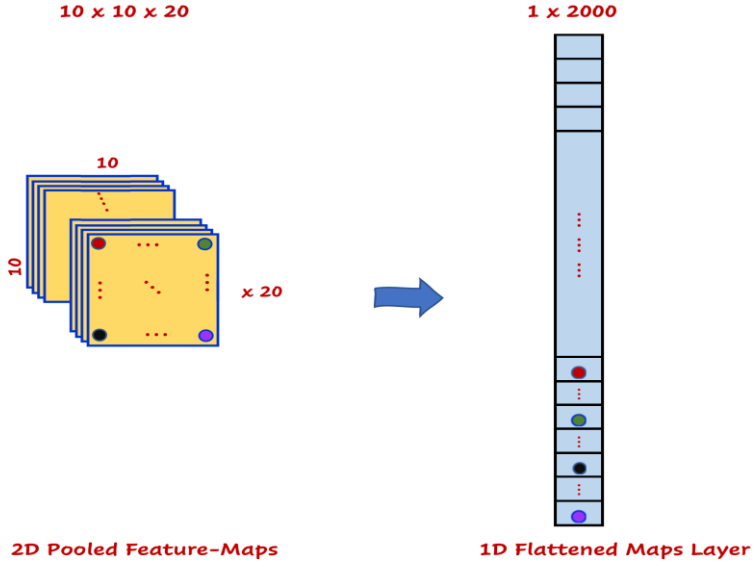


Figure 13. Implementation of flattening layer of our CNN.

Fully Connected Layer with ReLU Function: Fully Connected (FC) layers- as name implies- are those layers where all the inputs from one layer are connected to every activation unit of the next layer [43].

Commonly, FC layers are located as the last few layers of any CNN. Therefore, this module is responsible to compile the high level features extracted by previous layers (convolutional and pooling layers) into a reduced form of low level features in which they can be used by the classifier located at the output layer to provide classification probabilities. In this research, we have developed the FC layer with 200 neurons connected with 2000 nodes of the flattened (FL) layer which provide a layer complexity reduction by$10:1$. As the inputs pass from the units of FL layer through the neurons of FC layer, their values are multiplied by the weights and then pass into the employed activation function (normally ReLU function) just in the same way as in a the classical NN (i.e. shallow NN). Thereafter, they are forwarded to the output classification layer where each neuron expresses a class label. Note that, FC layer also goes through a backpropagation [43] process to determine the most accurate values of its trainable parameters ($weights\ W_{FL}\ x\ W_{FC}\ = 2000\ x\ 200$). Figure 14 illustrates the development for FC layer of our CNN.

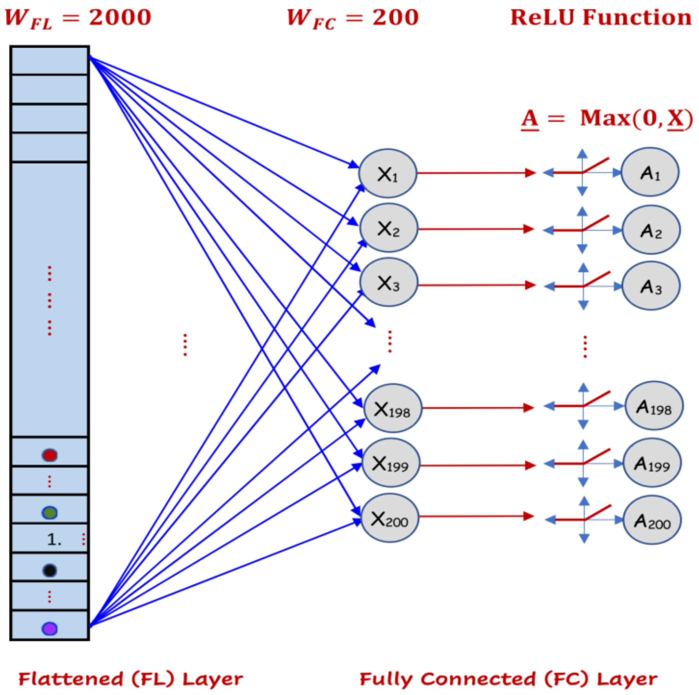


Figure 14. Implementation of flattening layer of our CNN.

Output Layer with SoftMax Function: This module is responsible to provide/predict the correct classification for each evaluated sample record of the utilized IoT attacks-dataset. Here we are providing two types of classification including the binary-classifier (normal or anomaly) and the multi-classifier (normal, DoS, Probe, R2L, U2R). The data points received from the 200 neurons of the FC layer ($A_1, A_2, \dots, A_{200}$) are fully connected with the five neurons (C_1, C_2, C_3, C_4, C_5) of the output classes ($j\ =\ 5\ vectors$) through the transposed weight connections (W_j^T). This is illustrated in Figure 15 and can be achieved algebraically as follows:

$$\underline{C} = \underline{W_j^T} . \underline{A} = \begin{bmatrix} \underline{W_1^T} \\ \underline{W_2^T} \\ \underline{W_3^T} \\ \underline{W_4^T} \\ \underline{W_5^T} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ \vdots \\ \vdots \\ \vdots \\ A_{198} \\ A_{199} \\ A_{200} \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \end{bmatrix}\ \ Where: \underline{W_1^T}, \underline{W_2^T}, \underline{W_3^T}, \underline{W_4^T}, \underline{W_5^T}\ are\ vectors\ of\ 1\ x\ 200 \qquad (5)$$

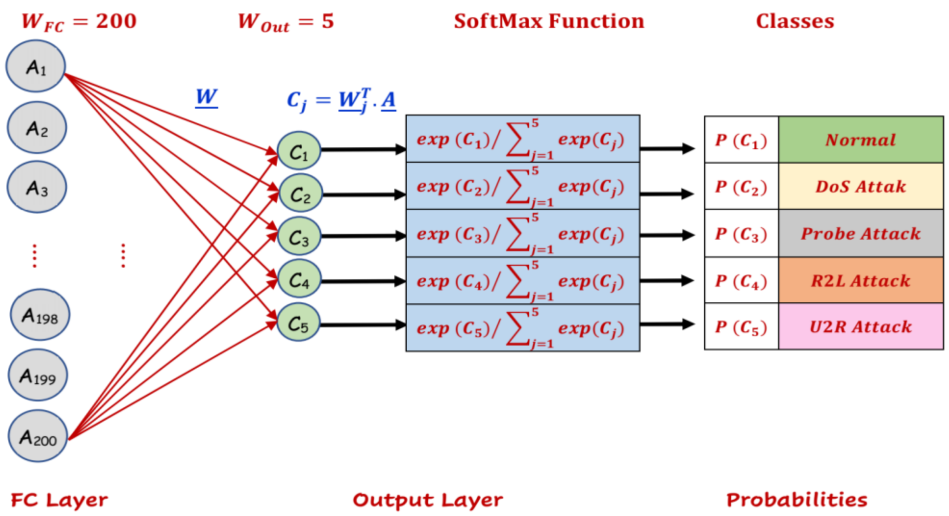


Figure 15. Implementation of the output layer with ***SoftMax*** of our CNN.

Note that, the output layer also goes through a backpropagation process to determine the most accurate values of its trainable parameters ($weights\ \ W_{FC}\ x\ W_{out}\ = 200\ x\ 5$). The last layer of the neural network is a $Softmax$ layer which has similar number of nodes as the output layer. $Softmax$ normalizes the output into a probability distribution on classes [43]. Specifically, $Softmax$ assigns numerical probability values for every class at the output layer where these probabilities should sum up to 1.0 (following a probability distribution). Given an input a vector (x) of (K) real numbers and (i) defines the index for the input values, then, SoftMax function $\sigma: \mathbb{R}^k \longmapsto \mathbb{R}^k$ is defined as follows:

$$\sigma(x)_i = e^{x_i}/\sum_{j=1}^{K} e^{x_i} \qquad for\ i\ = 1,2,3,\dots,K\ and\ x\ = (x_1,x_1,\dots,x_K) \in \mathbb{R}^k \tag{6}$$

For example, $Softmax$ might produce the following probabilities for an attack record:

	Multi-Classes Dataset				
	Normal	DoS	Probe	**R2L**	U2R
Label	1	2	3	4	5
Probability	0.001	0.040	0.008	0.950	0.001

3.4 System Integration

In this section, we integrate all the aforementioned subsystems and modules by Putting-It-All-Together to come up with complete system architecture of our IoT-IDCS-CNN. Figure 16 illustrates the top view architecture of the integrated system as a feedforward $CovNet$ network based IoT attack detection system.

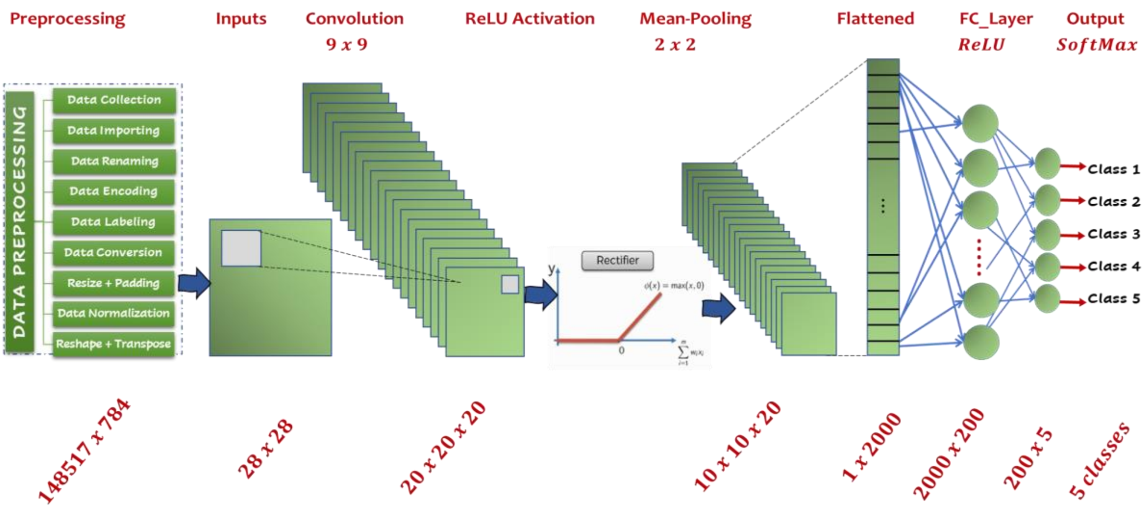


Figure 16. Top view architecture of the proposed IoT-IDCS-CNN.

According to the system architecture, after data preprocessing stages and using the $28\ x\ 28$ input matrix, we constructed 784 $(= 28\ x\ 28)$ input nodes. To extract features of the input data, the network encompasses a deep convolutional layer involving a depth of 20 convolution filters of size $(9\ x\ 9)$. Thereafter, the results of the convolutional layer pass via ReLU activation function which followed by the subsampling operation of the pooling layer. The pooling layer utilizes the average pooling method with $2\ x\ 2$ submatrices. The pooled features are then flattened to 2000 nodes. The classification/detection neural network comprises the single hidden fully connected (FC) layer and the output classification layer. This FC layer comprises 200 nodes along with ReLU activation function. Since our system requires the classification of the data into 5 classes, therefore, the output layer is implemented with 5 nodes with SoftMax activation function. The next table, Table 6, recaps the final integrated $CovNet$ based system for IoT attacks detection.

Table 6. Summary of the developed ***CovNet*** for IoT attacks detection/classification system.

Layer	Comment	Trainable Parameters
Preprocessing	148517 Sample each (28x28)	-
Input	28 x 28 nodes (784 nodes)	-
Convolution	20 convolution filters (9 x 9)+ReLU	W_{Con}(9 x 9 x 20)
Pooling	Mean pooling (2 x 2)	-
Flattening	2000 nodes	-
Fully Connected	200 nodes + ReLU	W_{FCL}(2000 x 200)
Output	5 nodes (or 2 nodes) + SoftMax	W_{Out}(200 x 5)

Moreover, the life cycle for the packet traffic received at the IoT gateway is provided in Figure 17 below. The input layer takes the encoded features generated from FE subsystem in order to be trained at the CNN which update the training parameters and generate the least cost/loss value (error) with optimal accuracy. The output layer employs the SoftMax classifier which is used to classify the data using two classification techniques include: binary classification technique which provides two categories (normal vs anomaly) and the multi-classification technique which provides five categories (normal, DoS attack, Probe attack, R2L attack, U2R attack).

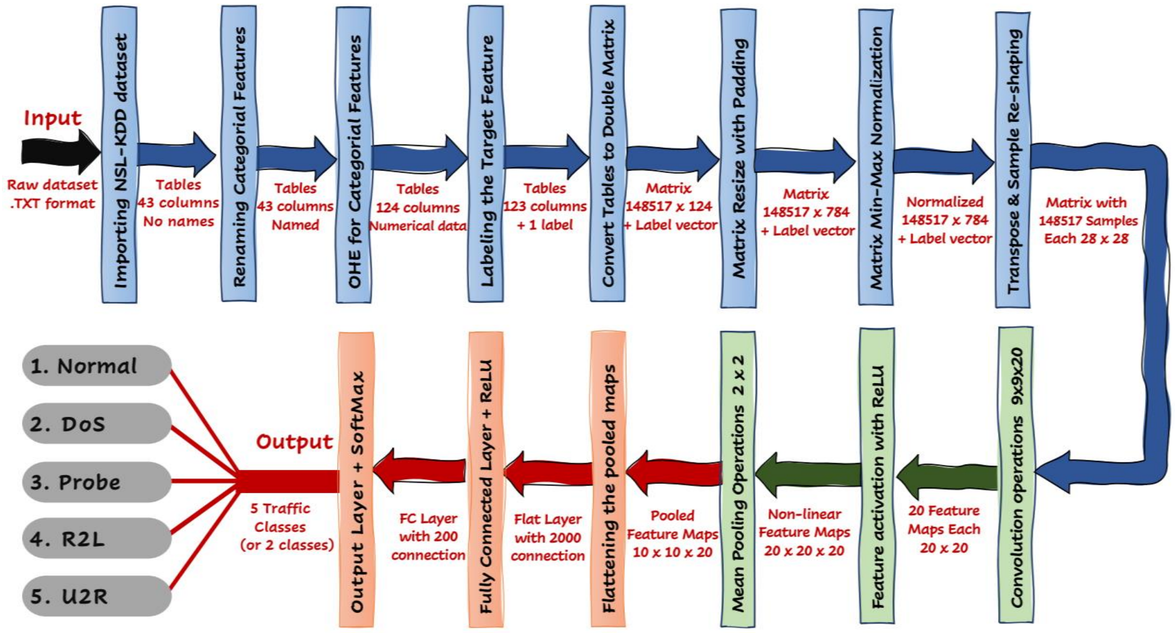


Figure 17. Comprehensive view of the Computation Process IoT-IDCS-CNN.

4. Simulation Environment

To implement, verify, and validate the proposed IoT attacks detection and classification system, the training and testing were performed on the NSL-KDD dataset involving the key attacks for IoT communication. The classifier model was determined to have either two classes (binary attack detection) or five classes (multi-attack classification). The proposed system was implemented in MATLAB 2019a. To evaluate the system performance, experiments were performed using a high-performance computing platform utilizing the power of central processing unit (CPU) and graphical processing unit (GPU) with Multicore structure of NVIDIA GeForce® Quadro P2000 Graphic card. The specifications for the workstation used in development, validation & verification are provided in Table 7.

Table 7. The system development and validation environment.

System Unit	Specifications
Processor Unit (CPU)	Intel Core I9-9900 CPU, 8 Cores, @ 4900 MHz
Graphics Card (GPU)	NVIDIA Quad P2000@1480 MHz, 5GB Mem, 1024 CUDA Cores
Cache Memory ($)	16 MB Cache @ 3192 MHz
Main Memory (RAM)	32 GB DDR4 @ 2666MHz
Operating System (OS)	64 bit, Windows 10 Pro.
Hard Disk Drive (HD)	SATA 1TB Drive + 256 GB SSD

Besides, the experimental setup for training/testing model has been configured as follows:

- ***Dataset Distribution:***
  - 85 % of the dataset used for training (i.e., ~ 128500 data sample records).
  - 15 % of the dataset used for testing (i.e., ~ 20000 data sample records).
- ***CovNet Configurations:***
  - Input (Sample) Size = 28 x 28.
  - Conv. Kernel Size = 9 x 9.
  - Activation function = ReLU.
  - Number of Hidden Layers = 5.
  - Number of Kernels = 20.
  - Mean Pooling filter size = 2 x 2.
  - Classifier function= SoftMax.
  - Number of Output classes = 2 or 5.
- ***Model Optimization Configurations:***
  - Optimization Algorithm = Mini Batch Gradient Descent (find minimum loss).
  - Mini_batch_size = 50, Momentum factor (β) =0.95, learning rate (α)=0.05.
  - Momentum updates= $\mathrm{Mom}_{\mathrm{Con}}$[9 x 9 x 20], $\mathrm{Mom}_{\mathrm{FCL}}$[2000x200], $\mathrm{Mom}_{\mathrm{Out}}$[200x5].
  - All Momentum updates were initialized using ZEROS matrices (zeros (size)).
- ***Training Model Configurations:***
  - Training technique = back-propagation with momentum (to update weights).
  - Trainable weights = $\mathrm{W}_{\mathrm{Con}}$[9 x 9 x 20] , $\mathrm{W}_{\mathrm{FCL}}$[2000x200], $\mathrm{W}_{\mathrm{Out}}$[200x5].
  - Backprop. Derivatives = $\mathrm{dW}_{\mathrm{Con}}$[9 x 9 x 20], d $\mathrm{W}_{\mathrm{FCL}}$[2000x200], $\mathrm{dW}_{\mathrm{Out}}$[200x5].
  - The number of epochs = 100 and the number of iterations per epoch = ~2500.
  - All trainable weights were initialized using random number generator (rand).
  - All backpropagation derivatives were initialized using ZEROS matrices.
- ***Weight update policy:***
  - $\mathrm{dW}_{\mathrm{Con}} = \mathrm{dW}_{\mathrm{Con}}/\mathrm{Mini\_batch\_size}$, $\mathrm{dW}_{\mathrm{FCL}} = \mathrm{dW}_{\mathrm{FCL}}/\mathrm{Mini\_batch\_size}$, $\mathrm{dW}_{\mathrm{Out}} = \mathrm{dW}_{\mathrm{Out}}/\mathrm{Mini\_batch\_size}$
  - $\mathrm{Mom}_{\mathrm{Con}} = \alpha * \mathrm{dW}_{\mathrm{Con}} + \beta * \mathrm{Mom}_{\mathrm{Con}};$ ➔ $\mathrm{W}_{\mathrm{Con}} = \mathrm{W}_{\mathrm{Con}} + \mathrm{Mom}_{\mathrm{Con}}$
  - $\mathrm{Mom}_{\mathrm{FCL}} = \alpha * \mathrm{dW}_{\mathrm{FCL}} + \beta * \mathrm{Mom}_{\mathrm{FCL}};$ ➔ $\mathrm{W}_{\mathrm{FCL}} = \mathrm{W}_{\mathrm{FCL}} + \mathrm{Mom}_{\mathrm{FCL}}$
  - $\mathrm{Mom}_{\mathrm{Out}} = \alpha * \mathrm{dW}_{\mathrm{Out}} + \beta * \mathrm{Mom}_{\mathrm{Out}};$ ➔ $\mathrm{W}_{\mathrm{Out}} = \mathrm{W}_{\mathrm{Out}} + \mathrm{Mom}_{\mathrm{Out}}$

5. Results and Discussion

Verification and validation (V&V) are essential activities and quality control factors that are performed independently to check the system compliance with requirements and specifications and that it fulfills its intended purpose. Typically, the verification process is defined as a number of activities used to examine the suitability of the system or component (i.e. are we building the product right). On the other hand, the validation process is defined as a number of activities used to examine the conformity of the system (or any of its elements) with its purpose and functions (i.e. are we building the right product). Note that while system validation is distinct from verification, however, the actions of both processes are integral and meant to be performed in coupling [45]. In this section, we provide a comprehensive verification and validation to check the system compliance with its intended objectives and purpose.

5.1 System Evlaution and Verification

To verify the effectiveness of the proposed system in compliance with its intended functionalities and missions, we have evaluated the system performance using the recommended testing dataset in terms of the classification accuracy, classification error percent and the classification time as follows:

$$Classification\ \ Accuracy\ (\%) = \frac{Correctly\ Predicted\ Samples}{Number\ of\ Testing\ Samples} \times 100\% \tag{7}$$

$$Classification\ Error\ (\%) \quad = \frac{Incorrectly\ Predicted\ Samples}{Number\ of\ Testing\ Samples} \times 100\% \tag{8}$$

$$Classification\ Time\ (ms) \quad = \sum_{i=1}^{No.\ Runs} Execution\ time\ (i) \times \frac{1000}{No.\ Runs} \tag{9}$$

The plot for the overall testing classification accuracy and overall classification loss (classification error) comparing the performance of the binary-classifier (2-Classes) and the multi-classifier (5-Classes) obtained during the validation process of NSL-KDD dataset are illustrated in Figure 18. According to the figure, at the beginning and after one complete pass (epoch) of testing process, both classifiers showed relatively low classification accuracy proportions with 85% and 79% registered for 2-Class classifier and 5-Class classifier, respectively. Thereafter, both classification accuracy curves begin to roughly be increasing in a stable tendency while testing epochs proceeds with faster and higher ceiling level obtained for the classification accuracy of 2-Class classifier. After training the system for 100 epochs, the system was able to record an overall testing accuracy proportions of 99.3% and 98.2% for 2-Classs classifier and 5-Classs classifier, respectively, for the given testing dataset samples. Conversely, it can be clearly seen that both classifiers showed relatively high classification error proportions at the beginning of the testing process with 15% and 21% registered for 2-Class classifier and 5-Class classifier after one testing epoch, respectively. Thereafter, both classification error rates started to systematically decline while the binary classifier progresses with faster threshold achieving 0.7% of incorrect prediction proportion (classification error percentage). However, the classification error rate proportion for the multi-classifier has saturated with less than 2.0% of incorrect prediction. This range of classification error of both classifiers (0.7% - 1.8%) is permitted to avoid underfitting or overfitting from the training loss (~0.0%) and training accuracy (~100%) and thus provided high-accuracy classification performance.

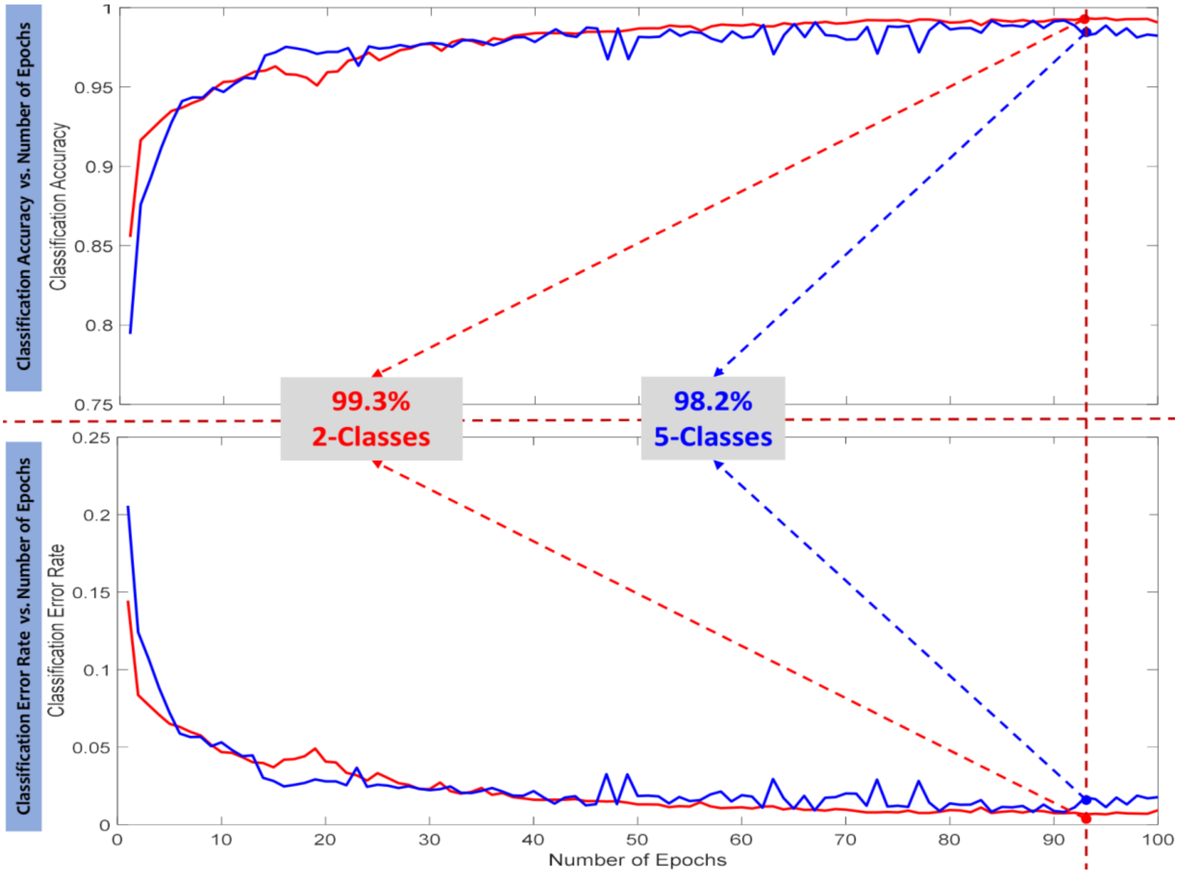


Figure 18. Testing Detection/Classification Accuracy/Error Rate vs. Number of Epochs.

Moreover, we have analyzed the time required to perform attack detection or classification for one IoT traffic sample. To obtain accurate and precise results, we have run the validation test for 500 times and then computed the time statistics for detection and classification. Figure 19 shows the detection/classification time performance for the proposed model (either 2-Class or 5-class classifier). According to the figure, the time required to detect/classify one sample record ranges from $(Min \approx 0.5662\ ms)$ to $(Max \approx 2.099\ ms)$ with average time of $(Mean \approx 0.9439\ ms)$ recorded for the 500 simulation runs. This average time $(around\ 1\ ms)$ is very useful for the system to run in dynamical environment such as the real time IDS applications.

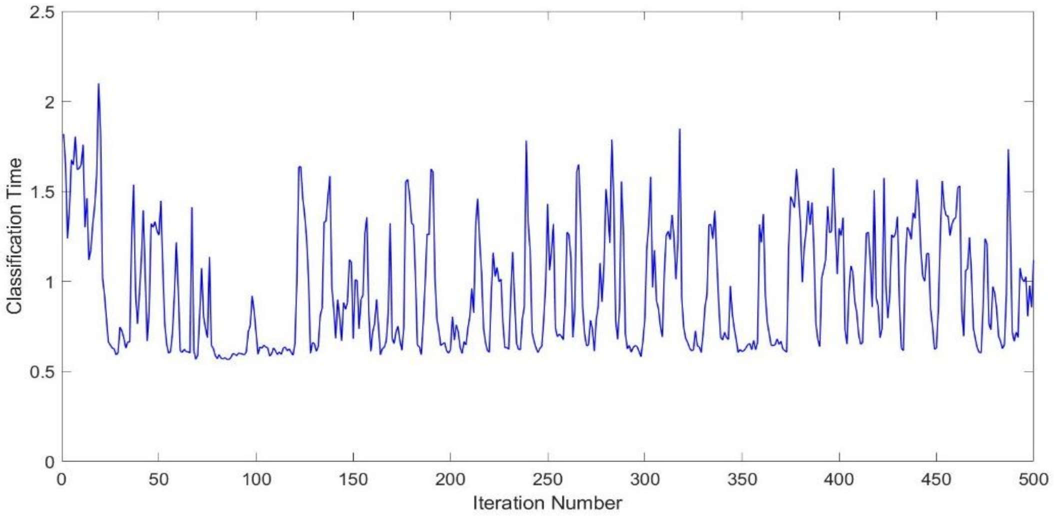
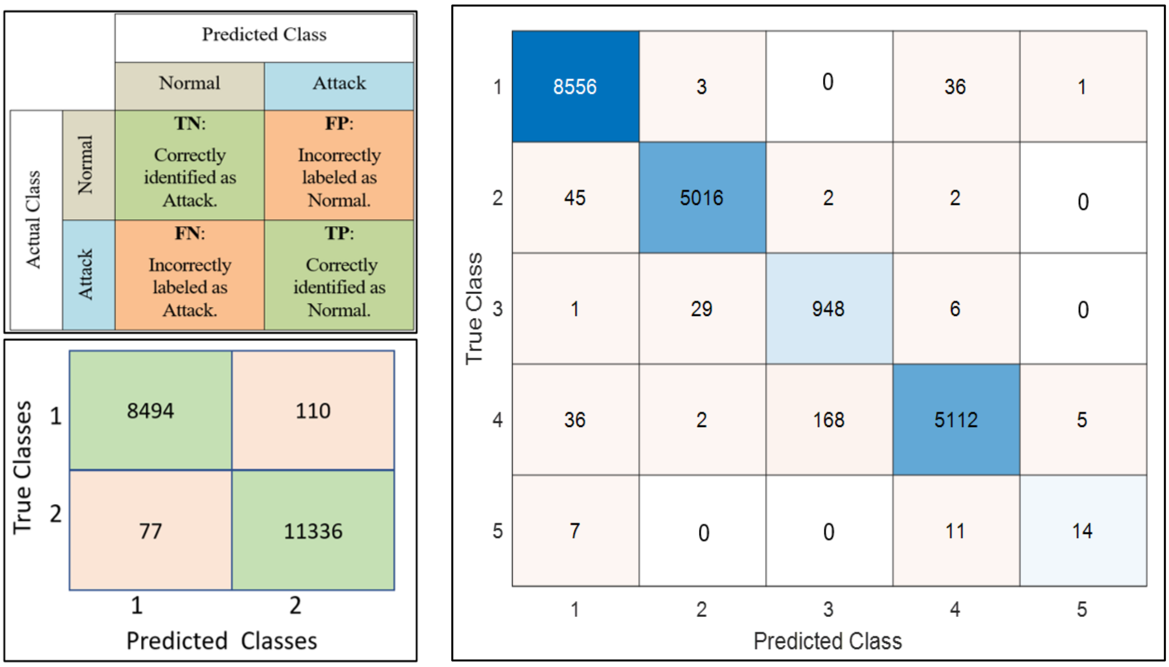


Figure 19. Run time performance of IoT Traffic classification over 500 simulation runs.

Furthermore, even though the classification accuracy measurement is the key significant factor used to evaluate the efficiency of the classification or detection system, we have evaluated the validation (testing) dataset using a confusion matrix with clear identification of True Positive (TP), True Negative (TN), False Positives (FP) and False Negatives (FN) analysis to provide more insight about the performance of the proposed. Figure 20 shows the general confusion matrix of our system, confusion Matrix results for 2-Class Classifier using the testing dataset, and the confusion matrix results for 5-Class Classifier using the testing dataset.



		Predicted Class	
		Normal	Attack
Actual Class	Normal	**TN**: Correctly identified as Attack.	**FP**: Incorrectly labeled as Normal.
	Attack	**FN**: Incorrectly labeled as Attack.	**TP**: Correctly identified as Normal.

True Classes	Predicted Classes 1	Predicted Classes 2
1	8494	110
2	77	11336

True Class	Predicted Class 1	2	3	4	5
1	8556	3	0	36	1
2	45	5016	2	2	0
3	1	29	948	6	0
4	36	2	168	5112	5
5	7	0	0	11	14

Figure 20. Confusion Matrix Analysis for both classification models.

Therefore, the confusion matrix parameters (i.e., TN, TP, FN, FP) can be used to compute some other performance evaluation metrics (has less importance than the accuracy metric) including: (a) the classification precision (detection rate) which is defined as the percentage of relevant instances (e.g. attacks) among the retrieved instances, (b) the classification recall (sensitivity) which is defined as the percentage of positive instances that are correctly labeled, (c) F1-Score which is defined as the average score involving precision and recall (i.e., utilizes both false negative and false positive), and (d) False alarm rate which is defined as the percentage of misclassified normal instances detected by the system [48]. These metrics can be calculated in the following equations while Table 8 summarizes the results of the overall evaluation metrics for our proposed system.

$$Precision \;=\; \frac{TP}{TP+FP} \times 100\% \tag{10}$$

$$Recall \;=\; \frac{TP}{TP+FN} \times 100\% \tag{11}$$

$$F - Score = 2\,x\,\frac{Recall\ x\ Precision}{Recall + Precision} \times 100\% \tag{12}$$

$$False\ Alarm\ Rate = \frac{FP}{TN+FP} \times 100\% \tag{13}$$

Table8. Summary of the overall evaluation metrics results

	2-Class Classification	5-Class Classification
Correctly predicted samples	19860	19640
Incorrectly predicted samples	140	360
Classification Accuracy	99.3%	98.2%
Classification Error Rate	00.7%	01.8%
Classification Precision	99.04%	98.27%
Classification Recall	99.33%	98.23%
F-Score Metric	99.18%	98.22%
False Alarm Rate (FAR)	01.28%	1.73%
Average Classification time	0.9246	0.9439

5.2 System Validation and Benchmarking

To validate the proficiency of proposed system in compliance with system purpose and specifications. To ensure high level of reliability of our system validation stage, we have conducted a 5-fold cross-validation process [47] that encompasses 5 different experiments for each classification model (total of 10 experiments) with different sets for training (~128,000 sample) and validation (20,000 sample) nominated for each experiments as demonstrated in Figure 21 which shows the distribution of the dataset across the folds for each conducted experiment.

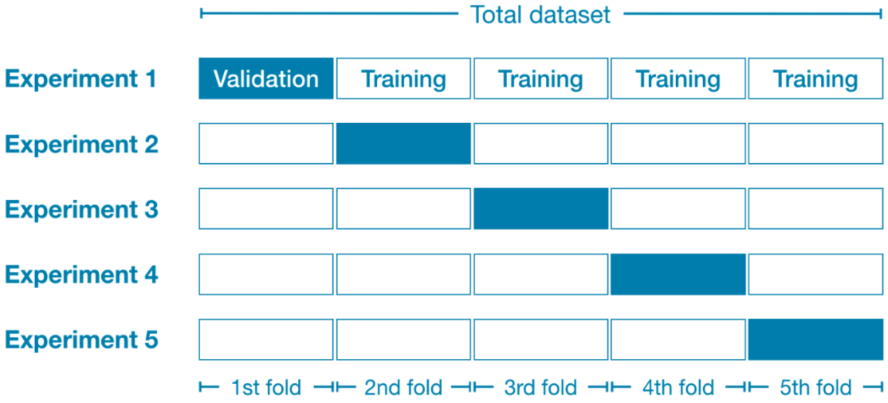


Figure 21. Scheme for 5-fold cross-validation of the proposed system .

For each experiment, we have evaluated the validation accuracy and validation error for the classification system models (2-Classes/5-Classes). Thereafter, the results obtained from the five experiments are averaged to provide an overall validation accuracy and validation error values. Consequently, the proposed system provided high level of stability and reliability across the dataset folds which confirm the system robustness in the mission of attacks detection and classification for IoT communications. The results of the 5-fold cross-validation are provided in Table 9 below.

Table 9. The results of 5-fold cross-validation of both classifiers (accuracy and error)

	2-Class		**5-Class**	
	Accuracy	**Error**	**Accuracy**	**Error**
Experiment 1	0.9930	0.0070	0.9820	0.0180
Experiment 2	0.9942	0.0058	0.9950	0.0050
Experiment 3	0.98750	0.01250	0.9907	0.0093
Experiment 4	0.99440	0.00560	0.9929	0.0071
Experiment 5	0.99320	0.00680	0.9966	0.0034
AVERAGE	**99.25%**	**0.75%**	**99.14%**	**0.86%**

Additionally, to gain more insight on the advantage of the proposed method, we benchmark IoT-IDS-CNN classification system by comparing its performance with other state-of-art machine learning based intrusion/attacks detection systems in terms in terms of classification accuracy metric.

For better and more reasonable evaluation, we have selected the related researches that employs machine learning techniques for intrusion/attacks detection/classification for the NSL-KDD dataset (the same used by our system) to be compared with our proposed IoT-IDS-CNN. Therefore, we summarize the classification accuracy metric values for related state-of-art research in the following table, Table 10, in chronological order. Accordingly, it can be obviously noticed, that the proposed IoT-IDS-CNN model has improved the cyber-attacks classification accuracy of other ML-IDS models by an improvement factor (IF) of (~1.03 – 1.25).

Table 10. Comparison with State-of-Art ML-IDS Employing Same Dataset.

Research	Data	Accuracy	IF %
K. Taher et. al. 2019 [16]	NSL-KDD Dataset	≈ 83.7%	117.3%
X. Gao et. al. 2019 [17]	NSL-KDD Dataset	≈ 85.2%	115.2%
S. Sapre, et. al. 2019 [18]	NSL-KDD Dataset	≈ 78.5%	125.1%
Chowdhry, 2017 [19]	NSL-KDD Dataset	≈ 94.6%	103.8%
Javaid et. al. 2016 [20]	NSL-KDD Dataset	≈ 88.4%	112.3%
Yadigar, et. al. 2016 [21]	NSL-KDD Dataset	≈ 91.7%	108.0%
Proposed Method	NSL-KDD Dataset	≈ 98.2~99.3%	____

Finally, although the other existing related researches for machine learning based intrusion/attack detection/classification use different cyber-attacks datasets, learning policies, programming techniques, and computing platforms, we still can compare the classification system performance in terms of testing accuracy metrics and the level of complexity for the developed method. Therefore, for better readability, we summarize the classification accuracy metrics for the other related state-of-art research in the following table, Table 11, in chronological order. According to the comparison of the table, it can be seen that the proposed approach produces attractive results in terms of classification accuracy showing superiority over all other compared methods.

Table 11. Comparison with State-of-Art ML-IDS Employing Different Dataset

Research	Data	Accuracy	IF %
S.Jan et.al 2019 [23]	CICIDS dataset	≈ 93.0%	106.7%
Roopak et. al. 2019 [24]	CICIDS Dataset	≈ 92.0%	107.9%
Ioannou et. al 2019 [25]	Simulated Dataset	≈ 81.0%	122.5%
Brun et al, 2018 [26]	Real-Time Dataset	≈ 75.0%	132.4%
Thing et. al 2017 [27]	AWID Dataset	≈ 98.0%	101.3%
Shukla et. al 2017 [28]	Simulated Dataset	≈ 75.0%	132.4%
Hodo et. al 2016 [29]	DoS Dataset	≈ 99.0%	100.3%
Kolias et. al 2016 [30]	AWID Dataset	≈ 92.0%	107.9%
Y. Li et. al. 2015 [31]	KDDCUP Dataset	≈ 92.0%	107.9%
Proposed Method	NSL-KDD Dataset	≈ 98.2~99.3%	____

6. Conclusions and Future Directions

An efficient and intelligent deep learning-based detection and classification system for cyberattacks in IoT communication networks (IoT-IDCS-CNN) was proposed, developed, tested, and validated in this paper. The proposed IoT-IDCS-CNN makes use of the high-performance computing employing the robust Nvidia GPUs (Quad-Cores, CUDA based) and the parallel processing employing the high-speed Intel CPUs (N-Cores, I9 based). For the purpose of system development, the proposed IoT-IDCS-CNN was decomposed into three subsystems including the Feature Engineering (FE) subsystem, the Feature Learning (FL) subsystem and the Detection and Classification (DC) subsystem. All subsystems were individually developed then, integrated, verified, and validated in this research. Because of the use of CNN based design, the proposed system was able to detect and classify the slightly mutated cyberattacks of IoT networks (represented collectively by NSL-KDD dataset which includes all the key attacks in the IoT computing) with detection accuracy of 99.3% of normal or anomaly traffic, and classify the IoT traffic into five categories with classification accuracy of 98.2%. Also, to ensure high level of reliability for system validation stage, we have conducted a 5-fold cross-validation process that encompasses 5 different experiments for each classification model. Moreover, and to provide more insight about the performance of the system, the proposed system was evaluated using the confusion matrix parameters (i.e., TN, TP, FN, FP) and computed some other performance evaluation metrics including: the classification precision, the classification recall, the F1-score of classification, and the false alarm rate. Eventually, the experimental evaluation results of IoT-IDCS-CNN system surpassed the results of many recent existing IDCS systems in the same area of study. Several recommendations for future research works may be considered to extend this study. These further recommendations include:

a) Additional data collection by setting up a real-time IoT communication network with sufficient number of nodes and gateways, incorporating nodes diversity. A future researcher can develop a new software system that catch and investigate any data packet communicated through the IoT environment (in-going and out-going) and come up with attacks to update an existing dataset or to come up with a new dataset. Note that the packet collection and investigation should be performed for a sufficient amount of time to provide more insights on the type of packets (normal or anomaly) processed at IoT networking. This can provide different perceptions of the operation of the device such as the utilization of processing unit, memory unit and the communication traffic. The collected data can be then deemed as normal or anomaly based on their behavior. For example, the normal data is related to the imitation of usual actions of local IoT devices, such as surveillance cameras. The anomaly data concerns with botnets/probes actions such as the communication with command & control units. At the end, the data can be labeled accordingly.
b) The proposed IoT-IDCS-CNN can be customized and used for intrusion detection incorporating other cyberattacks datasets such as AWID Dataset [49], CICIDS Dataset [50], DDoS dataset [51], UNSW-NB15 dataset [52] and others. This can be achieved by customizing the preprocessing and output layers accordingly with fine-tuning for the hidden layers as well as the model parameters and hyperparameters to obtain the maximum classification accuracy and least error rate.
c) The proposed IoT-IDCS-CNN can also be tuned and used to perform other real-life applications requiring image recognition and classification such as medical, biomedical, handwritten recognition applications and others.
d) Finally, the proposed system can be employed by IoT gateway device to provide intrusion detection services for a network of IoT devices such as a network of ARM Cortex based nodes. More investigation on the proposed IoT-IDCS-CNN can be reported including power consumption, memory utilizations, communication, and computation complexity over low power IoT nodes with tiny system components (such as the battery-operated/energy aware devices).

Author Contributions: Conceptualization, Q. A. Al-Haija; methodology, Q. A. Al-Haija; software, Q. A. Al-Haija; validation, Q. A. Al-Haija, formal analysis, Q. A. Al-Haija and C.D. McCurry; investigation, Q. A. Al-Haija and S. Z. Sabatto; resources, C. D. McCurry and S. Z. Sabatto; data curation, Q. A. Al-Haija; writing—original draft preparation, Q. A. Al-Haija; writing—review and editing, C. D. McCurry and S. Z. Sabatto; visualization, Q. A. Al-Haija; supervision, S. Z. Sabatto; project administration, C. D. McCurry; funding acquisition, C. D. McCurry. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded under the National Science Foundation Target Infusion Project (NSF-TIP) Program; titled “Targeted Infusion Project: Academic Enhancement of Electrical & Computer Engineering Program at Tennessee State University through IoT Research and Integrated Learning Environment” Award #: 1912313, funding period 2019-2022

Acknowledgments: Authors would like to thank the Department of Electrical and Computer Engineering in the College of Engineering at Tennessee State University for its administrative and technical support of this research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Alrawais, A. Alhothaily, C. Hu, and X. Cheng. Fog Computing for the Internet of Things: Security and Privacy Issues. *IEEE Internet Computing*, **2017**, *vol. 21(2)*, pp. 34-42. DOI: 10.1109/MIC.2017.37.
2. F. Chiti, R. Fantacci, M. Loreti and R. Pugliese. Context-aware wireless mobile automatic computing and communications: research trends & emerging applications. *IEEE Wireless Communications*, **2016**, *vol. 23(2):* pp. 86-92. DOI: 10.1109/MWC.2016.7462489
3. N. Silva, M. Khan, and K. Han. Internet of Things: A Comprehensive Review of Enabling Technologies, Architecture, and Challenges. *IETE Technical Review*, **2017**, *vol. 35(2):* pp. 1-16. DOI: 10.1080/02564602.2016.1276416
4. R. Mahmoud, T. Yousuf, F. Aloul and I. Zualkernan. Internet of things (IoT) security: Current status, challenges and prospective measures. Proceedings of the 10th International Conference for Internet Technology and Secured Transactions (ICITST), London, 2015, pp. 336-341, DOI: 10.1109/ICITST.2015.7412116.
5. Q. Jing, A.V. Vasilakos, J. Wan, et al. Security of the Internet of Things: perspectives and challenges. *Wireless Network, Springer*, **2014**, *Vol. 20*, pp. 2481–2501. DOI: 10.1007/s11276-014-0761-7
6. J. Zhou, Z. Cao, X. Dong and A. V. Vasilakos. Security and Privacy for Cloud-Based IoT: Challenges. *IEEE Communications Magazine*, **2017**, *Vol. 55(1)*, pp. 26-33. DOI: 10.1109/MCOM.2017.1600363CM.
7. Z. Yan, P. Zhang, A.V. Vasilakos. A survey on trust management for Internet of Things. *Journal of network and computer applications*, **2014**, *Vol. 42*, pp. 120-134. DOI: 10.1016/j.jnca.2014.01.014
8. P. Porambage, M. Ylianttila, C. Schmitt, P. Kumar, A. Gurtov and A. V. Vasilakos. The quest for privacy in the internet of things. *IEEE Cloud Computing*, **2016**, *Vol 3 (2)*, pp. 36-45,2016. DOI: 10.1109/MCC.2016.28.
9. C. Lin, D. He, X. Huang, K.R. Choo, A.V. Vasilakos. BSeIn: A blockchain-based secure mutual authentication with fine-grained access control system for industry 4.0. *Journal of Network and Computer Applications*, **2018**, *Vol. 116*, pp. 42-52. DOI: 10.1016/j.jnca.2018.05.005
10. S. Jangirala, A. K. Das and A. V. Vasilakos. Designing Secure Lightweight Blockchain-Enabled RFID-Based Authentication Protocol for Supply Chains in 5G Mobile Edge Computing Environment. *IEEE Transactions on Industrial Informatics*, **2020**, *Vol. 16(11)*, pp. 7081-7093, DOI: 10.1109/TII.2019.2942389.
11. M. Wazid, A. K. Das, V. BhBat, A.V. Vasilakos. LAM-CIoT: Lightweight authentication mechanism in cloud-based IoT environment. *Journal of Network and Computer Applications*. **2020**, *Vol. 150*. DOI: 10.1016/j.jnca.2019.102496
12. M. Wazid, A. K. Das, N. Kumar, A.V. Vasilakos and J. J. P. C. Rodrigues. Design and Analysis of Secure Lightweight Remote User Authentication and Key Agreement Scheme in Internet of Drones Deployment. *IEEE Internet of Things Journal*, **2019**, *Vol. 6 (2)*, pp. 3572-3584. DOI: 10.1109/JIOT.2018.2888821.
13. M. Wazid, A.K. Das, N. Kumar, A.V. Vasilakos. Design of secure key management and user authentication scheme for fog computing services. *Future Generation Computer Systems*. **2019**, *Vol. 91*, pp. 475-492. DOI: 10.1016/j.future.2018.09.017.
14. M. B. Mollah, M. A. K. Azad and A. Vasilakos. Secure Data Sharing and Searching at the Edge of Cloud-Assisted Internet of Things. *IEEE Cloud Computing*, **2017**, *Vol. 4 (1)*, pp. 34-42. DOI: 10.1109/MCC.2017.9.
15. Paar, J. Pelzl. *Understanding Cryptography*. Springer-Verlag Berlin Heidelberg Publisher, Germany, 2010, pp. 1–87. DOI: 10.1007/978-3-642-04101-3.
16. G. Caspi. Introducing Deep Learning: Boosting Cybersecurity with An Artificial Brain. Informa Tech, Dark Reading, Analytics, http://www.darkreading.com/analytics. 2016.
17. K. A. Taher, B. M.Y. Jisan and M. M. Rahman. Network Intrusion Detection using Supervised Machine Learning Technique with Feature Selection. Proceedings of the International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST), Bangladesh, 2019, pp. 643-646, DOI: 10.1109/ICREST.2019.8644161.
18. X. Gao, C. Shan, C. Hu, Z. Niu and Z. Liu. An Adaptive Ensemble Machine Learning Model for Intrusion Detection. *IEEE Access*, **2019**, *vol. 7*, pp. 82512-82521, DOI: 10.1109/ACCESS.2019.2923640.
19. S. Sapre, P. Ahmadi, K. Islam. A Robust Comparison of the KDDCup99 and NSL-KDD IoT Network Intrusion Detection Datasets Through Various Machine Learning Algorithms. **2019**. arXiv:1912.13204v1 [cs.LG].
20. M. Chowdhury, et. al. A few-shot deep learning approach for improved intrusion detection. Proceedings of the IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), New York, 2017, pp. 456-462, 10.1109/UEMCON.2017.8249084.

21. Q. Niyaz, W. Sun, A.Y Javaid, and M. Alam. Deep Learning Approach for Network Intrusion Detection System. Proceedings of the ACM 9th EAI International Conference on Bio-inspired Information and Communications Technologies, New York, 2016, pp. 1-6. DOI: 10.4108/eai.3-12-2015.2262516
22. A. R. Yusof, et.al. Adaptive feature selection for denial of services (DoS) attack. Proceedings of the IEEE Conference on Application, Information and Network Security (AINS), Miri, 2017, pp. 81-84, 10.1109/AINS.2017.8270429.
23. S.U. Jan, S. Ahmed, V. Shakhov, I. Koo. Toward a Lightweight Intrusion Detection System for the Internet of Things. *IEEE Access*, **2019**, *Vol.7*, pp. 42450- 42471. DOI: 10.1109/ACCESS.2019.2907965.
24. M. Roopak, G. Yun Tian and J. Chambers. Deep Learning Models for Cyber Security in IoT Networks. Proceedings of the IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), USA, 2019, pp. 0452-0457. DOI: 10.1109/CCWC.2019.8666588.
25. C. Ioannou and V. Vassiliou. Classifying Security Attacks in IoT Networks Using Supervised Learning. Proceedings of the 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), Greece, 2019, pp. 652-658. DOI: 10.1109/DCOSS.2019.00118.
26. O. Brun, Y. Yin, and E. Gelenbe. Deep learning with dense random neural network for detecting attacks against IoT-connected home environments. *Procedia Computer. Sci.*, **2018**, *vol. 134*, pp. 458–463. DOI: 10.1016/j.procs.2018.07.183
27. V. L. L. Thing. IEEE 802.11 Network Anomaly Detection and Attack Classification: A Deep Learning Approach. Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC), USA, 2017, pp. 1-6. DOI: 10.1109/WCNC.2017.7925567.
28. P. Shukla. ML-IDS: A machine learning approach to detect wormhole attacks in Internet of Things. Proceedings of the Intelligent Systems Conference (IntelliSys), London, 2017, pp. 234-240. doi: 10.1109/IntelliSys.2017.8324298.
29. E. Hodo et al. Threat analysis of IoT networks using artificial neural network intrusion detection system. Proceedings of the International Symposium of Network Computer Communication (ISNCC), 2016. pp. 1–6. DOI: 10.1109/ISNCC.2016.7746067
30. C. Kolias, G. Kambourakis, A. Stavrou, and S. Gritzalis. Intrusion detection in 802.11 networks: Empirical evaluation of threats and a public dataset. *IEEE Communications Surveys and Tutorials*, **2016**. *Vol. 18 (1)*, pp. 184-208. DOI: 10.1109/COMST.2015.2402161
31. Y. Li, R. Ma, and R. Jiao. A hybrid malicious code detection method based on deep learning. *International Journal of Security and Its Applications*, 2015, *Vol. 9*, pp 205–216. DOI: 10.14257/ijseia.2015.9.5.21
32. A. Ozgur and H. Erdem. A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015. *PeerJ Preprints*, **2016**, *Vol. 4:e1954v1*. DOI: 10.7287/PEERJ.PREPRINTS.1954
33. S. Revathi, Dr. A. Malathi. A Detailed Analysis on NSL-KDD Dataset Using Various Machine Learning Techniques for Intrusion Detection. *International Journal of Engineering Research & Technology (IJERT*), **2013**, *Vol. 2(2)*, pp. 1848- 1853.
34. Canadian Institute for Cybersecurity (CIS). NSL-KDD Dataset. https://www.unb.ca/cic/datasets/nsl.html. Retrieved on. 2019.
35. S. J. Stolfo, Wei Fan, Wenke Lee, A. Prodromidis and P. K. Chan. Cost-based modeling for fraud and intrusion detection: results from the JAM project. Proceedings of the DARPA Information Survivability Conference and Exposition. DISCEX'00, Hilton Head, SC, USA, 2000, pp. 130-144 vol.2, DOI: 10.1109/DISCEX.2000.821515.
36. C. Kolias, G. Kambourakis, A. Stavrou and J. Voas. DDoS in the IoT: Mirai and Other Botnets. *Journal of Computers*, **2017**. *vol. 50 (7)*, pp. 80-84, DOI: 10.1109/MC.2017.201.
37. Ambedkar, V. Kishore Babu. Detection of Probe Attacks Using Machine Learning Techniques. *International Journal of Research Studies in Computer Science and Engineering (IJRSCSE)*, **2015**, *Vol.2(3)*, pp. 25-29.
38. P. Pongle and G. Chavan. A survey: Attacks on RPL and 6LoWPAN in IoT. Proceedings of the International Conference on Pervasive Computing (ICPC), Pune, 2015. pp. 1-6. DOI: 10.1109/PERVASIVE.2015.7087034.
39. Y. Bengio; A. Courville; P. Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. **2013**, *Vol.35 (8):* 1798–1828. arXiv:1206.5538. doi:10.1109/tpami.2013.50. PMID 23787338.
40. D.J. Sarkar. Understanding Feature Engineering. Towards Data Science. Medium. https://towardsdatascience.com/tagged/tds-feature-engineering. 2018.

41. J. Brownlee. A Gentle Introduction to Padding and Stride for Convolutional Neural Networks. Deep Learning for Computer Vision, Machine Learning Mastery. https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks. 2019.
42. Kalay. Preprocessing for Neural Networks - Normalization Techniques. Machine Learning, Github.IO. https://alfurka.github.io/2018-11-10-preprocessing-for-nn. 2018.
43. Fei-Fei Li. CS231n: Convolutional Neural Networks for Visual Recognition. Computer Science, Stanford University, http://cs231n.stanford.edu. 2019.
44. J. Brownlee. A Gentle Introduction to the Rectified Linear Unit (ReLU). Deep Learning for Computer Vision, Machine Learning Master. https://machinelearningmastery.com. 2019.
45. INCOSE. INCOSE Systems Engineering Handbook, version 3.2.2. San Diego, CA, USA: International Council on Systems Engineering (INCOSE), INCOSE-TP-2003-002-03.2.2, 2012.
46. P. Gupta. Cross-Validation in Machine Learning. Medium: Towards data science. https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f. 2017.
47. S. Narkhede. Understanding Confusion Matrix. Medium: Towards data science. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62. 2018.
48. Phill Kim. MATLAB Deep Learning with Machine Learning, Neural Networks and Artificial Intelligence. Apress, 2017.
49. C. Kolias, G. Kambourakis, S. Gritzalis. Attacks and countermeasures on 802.16: Analysis & assessment. *IEEE Communications Surveys Tuts*., **2013**, *Vol. 15(1)*, pp. 487–514. DOI: 10.1109/SURV.2012.021312.00138.
50. CICIDS Dataset. DS-0917: Intrusion Detection Evaluation Dataset. https://www.impactcybertrust.org/dataset_view?idDataset=917
51. DDoS Dataset. Distributed Denial of Service (DDoS) attack Evaluation Dataset. https://www.unb.ca/cic/datasets/ddos-2019.html
52. N. Moustafa and J. Slay. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). Proceedings of the Military Communications and Information Systems Conference (MilCIS), Canberra, ACT, 2015, pp. 1-6, DOI: 10.1109/MilCIS.2015.7348942.