Intelligent Self-reliant Cyber-Attacks Detection and Classification System for IoT Communication Using Deep Convolutional Neural Network

Qasem Abu Al-Haija(✉), Charles D. McCurry, and Saleh Zein-Sabatto

Department of Electrical and Computer Engineering, Tennessee State University, 3500 John A. Merritt Blvd, Nashville, TN 37209, USA
Qabualha@Tnstate.edu

Abstract. Internet of Things (IoT) is a promising profound technology with tremendous expansion and effect. However, IoT infrastructures are vulnerable to cyber-attacks due to the constraints in computation, storage, and communication capacity for the endpoint devices such as thermostat, home appliance, etc. It was reported that 99% of the cyber-attacks are developed by slightly mutating previously known attacks to generate a new attack tending to be handled as a benign traffic through the IoT network. In this research, we developed a new intelligent self-reliant system that can detect mutations of IoT cyber-attacks using deep convolutional neural network (CNN) leveraging the power of CUDA based Nvidia-Quad GPUs for parallel computation and processing. Specifically, 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 showed a superior attacks' classification accuracy over the state-of-art machine learning based intrusion detection systems employing similar dataset. The obtained results showed more than 99.3% and 98.2% of attacks' classification accuracy for both binary-class classifier (normal vs anomaly) and multi-class classifier (five categories) respectively. All development steps and testing and verification results of the developed system are reported in the paper.

Keywords: Artificial intelligence · 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 "thing" 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 processing and manufacturing and others. The basic IoT layered architecture is shown in Fig. 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.

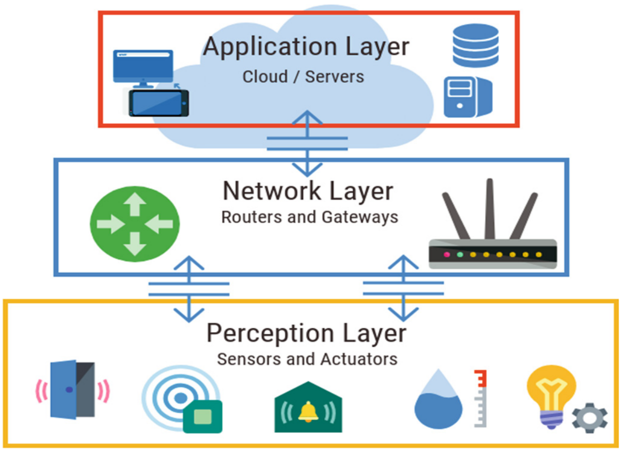


Fig. 1. IoT layered architecture [4].

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.) have constraints in computation, storage, and network capacity other than more complex endpoint devices, such as a smartphone, laptop etc., that may reside within the IoT infrastructure. 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 [5]. 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 them were developed by coupling the field machine learning techniques with cybersecurity field. It should be noted that, the majority of IoT attacks are developed as slight deviations (i.e. mutations) of earlier known cyberattacks [6]. Such slight mutations of these IoT attacks have been demonstrated to be difficult to identify/classify using traditional machine learning techniques. Indeed, few promising state-of-art researches were conducted for cybersecurity using deep neural networks models such as [7–12].

In this paper, we propose a new intelligent system that can detect mutations of common IoT cyber-attacks using non-traditional machine learning techniques exploiting the power of Nvidia-Quad GPUs. The proposed system employs the convolutional neural network (CNN) along its associated machine learning algorithms to classify the NSL-KDD dataset records (we call our system as 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:

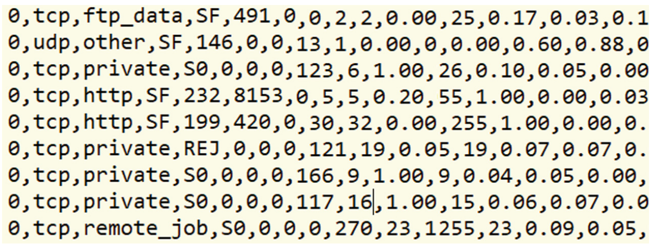
- We provide a comprehensive efficient detection/classification model that can classify the IoT traffic records of NSL-KDD dataset into two (Binary-Classifier) or five (Multi-Classifier) classes. Also, we present detailed preprocessing operations for the collected dataset records prior to the use with deep learning algorithms. Besides, we illustrate a comprehensive view of the computation process of our IoT-IDCS-CNN.
- We provide a simplified 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) in addition to benchmarking of our findings with other related state-of-art works.

The rest of this paper is organized as follows: Sect. 2 introduces and justify the dataset of IoT cyber-attacks employed in our system. Section 3 provides details of the proposed system design and architecture. Section 4 presents the environment for system development and validation. Section 5 discusses the details about experimental evaluation, comparison, and discussion. Finally, Sect. 6 concludes the findings of paper.

2 Dataset of Cyber-Attacks

NSL-KDD [13] is a reduced version of the original KDD'99 dataset [14] 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, NSL-KDD dataset is collected and 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) [15], Probe (side channel) [16], Root to Local (R2L) [17], User to Root (U2R) [17].
(c) It is obtainable as TXT/CSV filetype consisting a reasonable number of non-redundant records in the train and test sets. This improves the classification process by avoid the bias towards more frequent records.



```
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,
```

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

(d) It reveals high-level of IoT traffic structures and cyberattacks as well as it can be customized, expanded, and regenerated [13].

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 [18]. The statistics of traffic distribution of NSL-KDD dataset is summarized in Table 1.

Table 1. Statistics of traffic distribution of NSL-KDD dataset [14].

	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 Fig. 3.

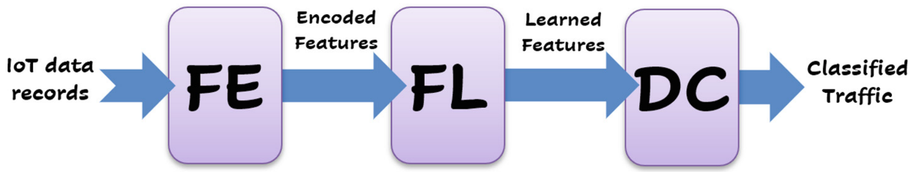


Fig. 3. The three main subsystems composing the proposed system.

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.

Renaming Categorial Features: Four of imported 43 features are categorial 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 categorial columns have been renamed accordingly in this stage.

One Hot Encoding of Categorial Features: This module is responsible for conversion of the categorial data records into numerical data records in order to be employed by the neural network. Therefore, three categorial features undergo through One Hot Encoding process (1-N encoding) [19]. These features are the protocol column, the service column, and the flag column. The class feature/column is left for samples labeling process.

Labeling the Target Feature: This stage concerns with the samples labeling using numerical (integer) labels for the target classes. Therefore, the categorial '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 categorial data are exist anymore). As a result of 1-N encoding and numerical labeling, we converted the dataset into 123 features and one data label.

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 (148517×124).

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 × 124 to the new size of 148517 × 784, since the input size of every individual sample processed at FL subsystem is 28 × 28 (=784). Thereafter, the new empty records of this matrix were padded with zero-padding technique [20]. Besides, to avoid any feature biasing in the samples of the dataset, the padded records were distributed equally around the data samples.

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) [21]. MX-Norm is well-known method to normalize data as its commonly used in machine learning applications. In this method, we scan all the all 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.

Converting Tables to 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 × 784) will be reshaped into two-dimensional matrix (28 × 28).

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 of various layers including convolution, activation, pooling, flatten and others. Convolutional layers are the core constructing 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 FL subsystem is responsible to develop the 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 updates 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 [22]. These feature-maps are produced by convolving (multiply and accumulate) the original matrix ($n_{in} \times n_{in}$) using a number (N) of ($k \times k$) convolution filters with padding size (p) and stride size of (s) which yields the feature maps ($n_{out} \times n_{out}$). The size of the resultant feature maps can be evaluated as follows:

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

In this research, we have applied 20 convolution filters (9 × 9) over the 28 × 28 input samples with (p = 0, and $s = 1$) which resulted in 20 feature map each (20 × 20).

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 [23].

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 [22]. 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 ($L_{out} \times L_{out}$), the pooling filter ($f \times f$) is independently applied over the rectified feature-maps ($L_{in} \times L_{in}$) with stride (s) as follows:

$$\boldsymbol{L_{out} = (L_{in} - f)/s + 1} \tag{2}$$

In this research, we have applied 20 pooling operation (2 × 2) over the 20 × 20 rectified feature-maps with ($s = 2$) which resulted in 20 feature map each (10 × 10).

3.3 Implementation of Detection and Classification (DC) Subsystem

DC subsystem is responsible of 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 [22]. 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 ($F_{in} \times F_{in}$) then the flattened output (F_{out}) can be obtained by linear multiplication of the input dimensions by the number of maps, that's it:

$$\boldsymbol{F_{out}} = \boldsymbol{N} \times \boldsymbol{F_{in}} \times \boldsymbol{F_{in}} \tag{3}$$

In this research, since we have 20 pooled feature maps ($N = 20$), each with dimension of 10×10 ($F_{in} = 10$), then, our flatten layer comprise of 2000 nodes.

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 [22]. 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 [24] process to determine the most accurate values of its trainable parameters ($weights\ W_{FL} \times W_{FC} = 2000 \times 200$).

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 ($A_1, A_2, \ldots, A_{200}$) 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 can be achieved algebraically as follows:

$$\underline{C} = \underline{W_j^T} \cdot \underline{A} = \begin{bmatrix} \underline{W_1^T} \\ \underline{W_2^T} \\ \underline{W_3^T} \\ \underline{W_4^T} \\ \underline{W_5^T} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ \vdots \\ \vdots \\ \vdots \\ A_{198} \\ A_{199} \\ A_{200} \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \end{bmatrix} \quad \textbf{\textit{Where}}: \underline{W_1^T}, \underline{W_2^T}, \underline{W_3^T}, \underline{W_4^T}, \underline{W_5^T} \textbf{\textit{ are vectors of }} \mathbf{1\ x\ 200} \tag{4}$$

Note that, the output layer also goes through a backpropagation process to determine the most accurate values of its trainable parameters ($weights\ W_{FC} \times W_{out} = 200 \times 5$). 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 [22]. 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 $\sigma : \mathbb{R}^k \mapsto \mathbb{R}^k$ is defined as follows:

$$\sigma(\boldsymbol{x})_i = e^{x_i} / \sum_{j=1}^{K} e^{x_i} - \boldsymbol{for}\, \boldsymbol{i} = 1, 2, 3, \ldots, \boldsymbol{K}\, \boldsymbol{and}\, \boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_K) \in \mathbb{R}^k \tag{5}$$

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 4 illustrates the top view architecture of the integrated system as a feedforward *CovNet* network based IoT attack detection system.

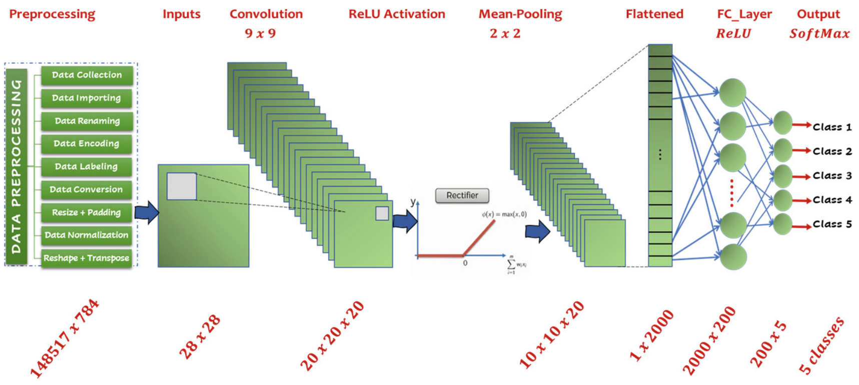


Fig. 4. Top view architecture of the proposed IoT-IDCS-CNN.

According to the system architecture, after data preprocessing stages and using the 28 × 28 input matrix, we constructed 784 (=28 × 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 × 9). Thereafter, the results of the convolutional layer passes via ReLU activation function which followed by the subsampling operation of the pooling layer. The pooling layer utilizes the average pooling method with 2 × 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 2, recaps the final integrated *CovNet* based system for IoT attacks detection.

Moreover, the life cycle for the packet traffic received at the IoT gateway is provided in Fig. 5 below. The input layer takes the encoded features generated from FE subsystem

Table 2. Summary of the developed CovNet for IoT attacks detection/classification system.

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

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).

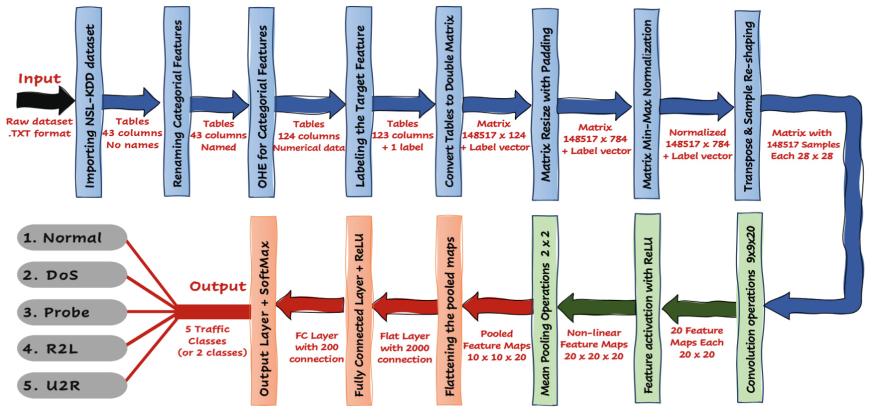

Input
Raw dataset .TXT format
Importing NSL-KDD dataset
Tables 43 columns No names
Renaming Categorial Features
Tables 43 columns Named
OHE for Categorial Features
Tables 124 columns Numerical data
Labeling the Target Feature
Tables 123 columns + 1 label
Convert Tables to Double Matrix
Matrix 148517 x 124 + Label vector
Matrix Resize with Padding
Matrix 148517 x 784 + Label vector
Matrix Min-Max Normalization
Normalized 148517 x 784 + Label vector
Transpose & Sample Re-shaping
Matrix with 148517 Samples Each 28 x 28
Convolution operations 9x9x20
20 Feature Maps Each 20 x 20
Feature activation with ReLU
Non-linear Feature Maps 20 x 20 x 20
Mean Pooling Operations 2 x 2
Pooled Feature Maps 10 x 10 x 20
Flattening the pooled maps
Flat Layer with 2000 connection
Fully Connected Layer + ReLU
FC Layer with 200 connection
Output Layer + SoftMax
Output
5 Traffic Classes (or 2 classes)
1. Normal
2. DoS
3. Probe
4. R2L
5. U2R

Fig. 5. 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 ($IoT - IDCS - CNN$), 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 via Table 3.

Table 3. The system develompment 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, 5 GB Mem, 1024 CUDA Cores
Cache Memory ($)	16 MB Cache @ 3192 MHz
Main Memory (RAM)	32 GB DDR4 @ 2666 MHz
Operating System (OS)	64 bit, Windows 10 Pro
Hard Disk Drive (HD)	SATA 1 TB 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 = $\text{Mom}_{Con}[9 \times 9 \times 20]$, $\text{Mom}_{FCL}[2000 \times 200]$, $\text{Mom}_{Out}[200 \times 5]$.
  - All Momentum updates were initialized using ZEROS matrices ($zeros(size)$).
- **Training Model Configurations:**
  - Training technique = back-propagation with momentum (to update weights).
  - Trainable weights = $W_{Con}[9 \times 9 \times 20]$, $W_{FCL}[2000 \times 200]$, $W_{Out}[200 \times 5]$.

- Backprop. Derivatives = $dW_{Con}[9 \times 9 \times 20]$, $dW_{FCL}[2000 \times 200]$, $dW_{Out}[200 \times 5]$.
- 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:**
  - $dW_{Con} = \frac{dW_{Con}}{Mini\_batch\_size}$, $dW_{FCL} = \frac{dW_{FCL}}{Mini\_batch\_size}$, $dW_{Out} = \frac{dW_{Out}}{Mini\_batch\_size}$
  - $\mathrm{Mom}_{Con} = \alpha * dW_{Con} + \beta * \mathrm{Mom}_{Con}; \rightarrow W_{Con} = W_{Con} + \mathrm{Mom}_{Con}$
  - $\mathrm{Mom}_{FCL} = \alpha * dW_{FCL} + \beta * \mathrm{Mom}_{FCL}; \rightarrow W_{FCL} = W_{FCL} + \mathrm{Mom}_{FCL}$
  - $\mathrm{Mom}_{Out} = \alpha * dW_{Out} + \beta * \mathrm{Mom}_{Out}; \rightarrow W_{Out} = W_{Out} + \mathrm{Mom}_{Out}$.

5 Results and Discussion

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:

$$\boldsymbol{ClassificationAccuracy(\%)} = \frac{\boldsymbol{CorrectlyPredictedSamples}}{\boldsymbol{NumberofTestingSamples}} \times 100\% \tag{6}$$

$$\boldsymbol{Classification\ Error(\%)} = \frac{\boldsymbol{Incorrectly\ Predicted\ Samples}}{\boldsymbol{Number\ of\ Testing\ Samples}} \times 100\% \tag{7}$$

$$\boldsymbol{Classification\ Time(ms)} = \sum_{\boldsymbol{i}=1}^{\boldsymbol{No.Runs}} \boldsymbol{Execution\ time\ (i)} \times \frac{1000}{\boldsymbol{No.Runs}} \tag{8}$$

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 Fig. 6. 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.

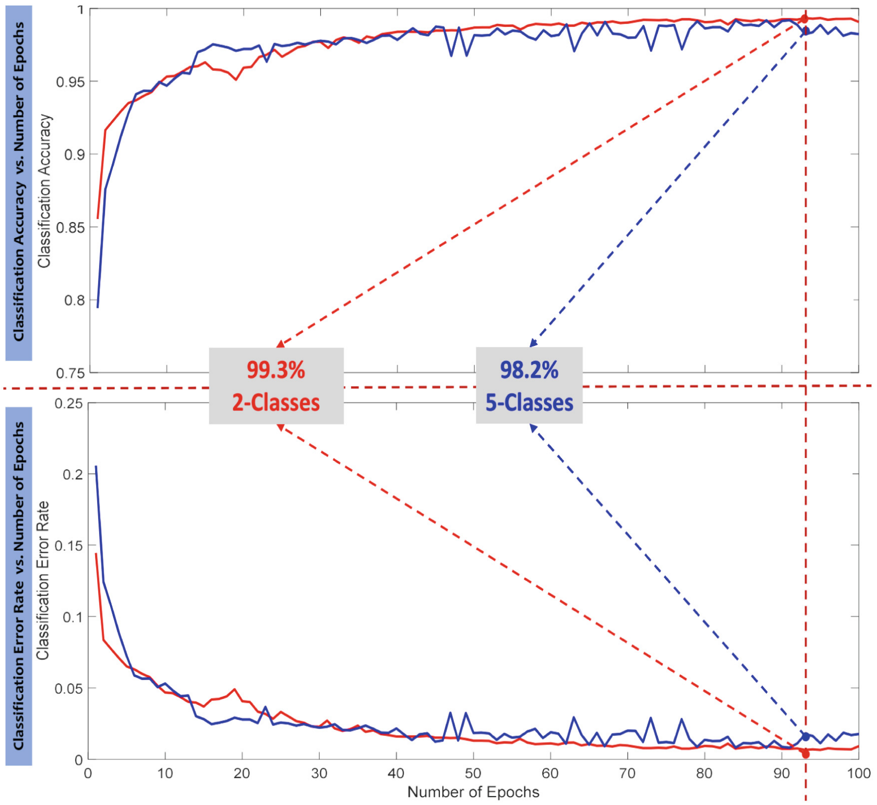


Fig. 6. Testing Detection/Classification Accuracy/Error Rate vs. number of Epochs.

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 7 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. Finally, 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's in the following table, Table 4, 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).

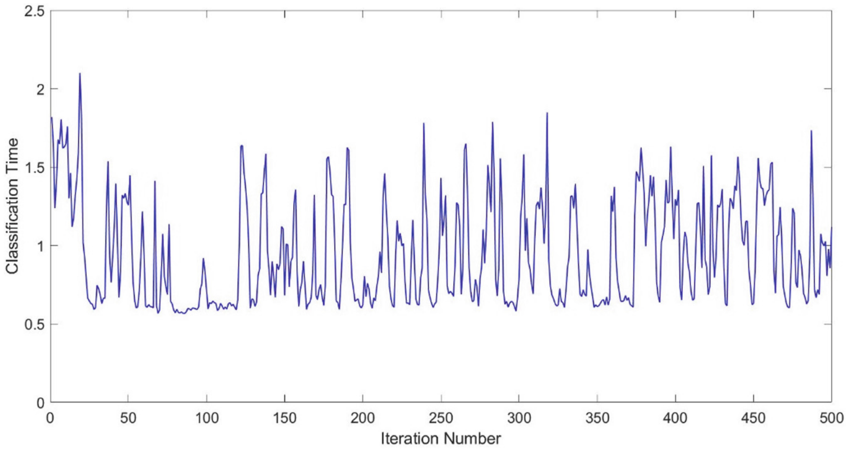


Fig. 7. Run time performance of IoT Traffic classification over 500 simulation runs.

Table 4. Comparison with state-of-art ML-IDS employing same dataset.

Research	Method	Accuracy	Remark	IF %
K. Taher et al. 2019 [6]	ANN with SVM Classifier	≈83.7%	3-classes, with 2 hidden layers and used only 35-features	1.173
X. Gao et al. 2019 [7]	Deep NN with ensemble voting	≈85.2%	5-classes, 3-methods: Decision Tree, Random Forest, K-Nearest	1.152
S. Sapre, et al. 2019 [8]	Different ML-IDS techniques	≈78.5%	5-classes, with 2 hidden layers and Naïve Bayes Classifier	1.251
Chowdhry, 2017 [9]	DNN with SVM + K-Nearest	≈94.6%	5-classes, 6 hidden layers, used only 35-features, 1-NN Classifier	1.038

(*continued*)

Table 4. (*continued*)

Research	Method	Accuracy	Remark		IF %
Javaid et al. 2016 [10]	Deep Autoencoders (DAE)	≈88.4%	2-classes, with 5 hidden layers and SoftMax classifier		1.123
Yadigar, et al. 2016 [11]	DNN with Extreme Learning Machine(ELM)	≈91.7%	2-classes, Hybrid feature selection: (consistency subset evaluation + DoS characteristic features), used only DoS attacks, 17 Features		1.080
Proposed Method	CNN: IoT-IDS_ *CNN*	≈98.2% ≈99.3%	2-classes 5-classes	CNN, 5 hidden layers and SoftMax classifier	-

6 Conclusions and Future Directions

A new intelligent and self-reliant intrusion detection system (IDS) for IoT cyber-attacks detection and classification using deep convolutional neural network (CNN) was proposed and developed in this paper. The proposed system takes advantage of the powerful CUDA based Nvidia-Quad GPUs and the Multi-Core I9 Intel CPUs of the system development platform for improved computation and faster processing. Specifically, the proposed system is composed of three subsystems: Feature Engineering (FE) subsystem, Feature Learning (FL) subsystem and Detection and Classification (DC) subsystem. All subsystems were completely developed, integrated, verified, and validated in this research. Because of the use of CNN based design, the proposed system was able to detect the slightly mutated attacks of IoT networking (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**%. Such results of the implemented system outperform several similar up-to-date state-of-art research results in the same area of study. In future, we will consider the inclusion of more detailed description of our system modules and the machine learning algorithms. Also, we will perform and incorporate more 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 the Precision, Recall, F-Score Metric, False Alarm Rate. Moreover, we will provide more robust validation for our system using K-Fold Cross Validation as well as the comparison of our system with other State-of-Art ML-IDS employing different dataset.

References

1. Alrawais, A., Alhothaily, A., Hu, C., Cheng, X.: Fog Computing for the Internet of Things: Security and Privacy Issues. IEEE Internet Comput. **21**(2), 34–42 (2017)
2. Chiti, F., et al.: Context-aware wireless mobile automatic computing and communications: research trends & emerging applications. IEEE Wirel. Commun. **23**(2), 86–92 (2016)

3. Silva, B.N., Khan, M., Han, K.: Internet of things: a comprehensive review of enabling technologies, architecture, and challenges. IETE Tech. Rev. **35**(2), 205–220 (2017). https://doi.org/10.1080/02564602.2016.1276416
4. Mahmoud, R. et. al. IoT security: current status. In: 10th International Conference for Internet Technology and Secured Transactions (ICITST), pp. 336 341 (2015)
5. Paar, C., Pelzl, J.: Understanding Cryptography. Springer, Berlin https://doi.org/10.1007/978-3-642-04101-3 (2010)
6. Caspi, G.: Introducing Deep Learning: Boosting Cybersecurity with An Artificial Brain. Informa Tech, Dark Reading, Analytics http://www.darkreading.com/analytics. (2016)
7. Taher, K.A., Jisan, B.M., and Rahman, M.M.: Network intrusion detection using supervised machine learning technique with feature selection. In: IEEE International Conference on Robotics, Electrical and Signal Processing Techniques, pp. 643–646 (2019)
8. Gao, X., Shan, C., Hu, C., Niu, Z., Liu, Z.: An adaptive ensemble machine learning model for intrusion detection. IEEE Access **7**, 512–521 (2019)
9. Sapre, S., Ahmadi, P., Islam, K.: A Robust Comparison of the KDDCup99 and NSL-KDD IoT Network Intrusion Detection Datasets Through Various Machine Learning Algorithms. arXiv:1912.13204v1 [cs.LG]. (2019)
10. Chowdhury, M. et. al.: A few-shot deep learning approach for improved intrusion detection. In: IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), New York, pp. 456–462, https://doi.org/10.1109/uemcon.2017.8249084. (2017)
11. Niyaz, Q., Sun, W., Javaid, A.Y., Alam, M.: Deep learning approach for network intrusion detection system. In: ACM 9th EAI International Conference on Bio-inspired Information and Communications Technologies, New York (2016)
12. Yusof, A.R. et.al.: Adaptive feature selection for denial of services (DoS) attack. In: IEEE Conference on Application, Information and Network Security (AINS), Miri 2017, pp. 81–84 https://doi.org/10.1109/ains.2017.8270429. (2017)
13. Canadian Institute for Cybersecurity (CIS). NSL-KDD Dataset. https://www.unb.ca/cic/datasets/nsl.html. Retrieved on. (2019)
14. A. Ozgur and H. Erdem. A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015. PeerJ Preprints 4:e1954v1. (2016)
15. C. Kolias, G. Kambourakis, A. Stavrou and J. Voas. DDoS in the IoT: Mirai and Other Botnets," Journal of Computers, vol. 50 (7), pp. 80–84, https://doi.org/10.1109/mc.2017.201. (2017)
16. C. Ambedkar, V. Kishore Babu. Detection of Probe Attacks Using Machine Learning Techniques. International Journal of Research Studies in Computer Science and Engineering (IJRSCSE), Vol2, No. 3, Pp 25–29. (2015)
17. P. Pongle and G. Chavan. A survey: Attacks on RPL and 6LoWPAN in IoT," 2015 International Conference on Pervasive Computing (ICPC), Pune, 2015, pp. 1–6. https://doi.org/10.1109/pervasive.2015.7087034. (2015)
18. Bengio, Y., Courville, A., Vincent, P.: Representation learning: a review and new perspectives. IEEE Trans. Pattern Anal. Mach. Intell. **35**(8), 1798–1828. arXiv:1206.5538. https://doi.org/10.1109/tpami.2013.50. PMID 23787338. (2013)
19. Sarkar, D.J.: Understanding Feature Engineering. Towards Data Science. Medium. https://towardsdatascience.com/tagged/tds-feature-engineering. (2018)
20. Brownlee, J.: 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)
21. Kalay, A.: Preprocessing for Neural Networks - Normalization Techniques. Machine Learning, Github.IO. https://alfurka.github.io/2018-11-10-preprocessing-for-nn. (2018)
22. Li, F.: CS231n: Convolutional Neural Networks for Visual Recognition. Computer Science, Stanford University, http://cs231n.stanford.edu. (2019)

23. Brownlee, J.: A Gentle Introduction to the Rectified Linear Unit (ReLU). Deep Learning for Computer Vision, Machine Learning Master. https://machinelearningmastery.com. (2019)
24. Kim, P.: MATLAB Deep Learning with Machine Learning. Apress, Neural Networks and Artificial Intelligence (2017)