Comparative Analysis of Deep Learning Techniques on LOT devices

Moushumi Barman[1] & Bobby Sharma[2†]

[1*]Department of CSE, Assam Don Bosco University, Azara, Guwahati, 781017, Assam, India
[2]Department of CSE, Assam Don Bosco University, Azara, Guwahati, 781017, Assam, India
*Corresponding author: **Moushumi Barman**
DOI: 10.54882/13202313202317348

Abstract

With the rapid proliferation of Internet of Things (IoT) devices, the threat landscape has expanded, posing significant challenges for secu-rity and privacy. Malware attacks targeting IoT devices have become a pressing concern, as they can compromise sensitive data, disrupt ser-vices, and even lead to physical harm. This research paper presents a comparative analysis of deep learning techniques for detecting mal-ware on IoT devices. The study focuses on addressing the unique challenges associated with limited resources, diverse communication protocols, and dynamic environments of IoT devices. A benchmark dataset comprising real-world IoT network traffic, encompassing benign and malicious activities, is utilized. Various deep learning models, including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Autoencoders (AE), Multilayer Perceptron (MLP), and Radial Basis Function Neural Networks (RBFNs), are implemented and trained on the dataset. Performance evaluation based on accu-racy, along with computational complexity and resource consumption, highlights the most effective techniques. The CNN model identifies malware patterns accurately by exploiting spatial dependencies, while RNNs capture temporal dependencies effectively. Autoencoders detect anomalies by reconstructing normal behavior. MLPs and RBFNs pro-vide additional insights into the dataset and potential attack vectors.

Keywords: Deep Learning, Internet of Things, CNN, Autoencoder, RNN

1 Introduction

"Internet" coined by Bob Kahn and Vint Cerf in ARPANET laboratories describes a broad array of protocols and applications that are constructed on highly developed and associated computing devices that are able to collaborate on resources and transmitting information, obliging worldwide spectators all the time. Further, the spectators grep their focus towards fusion of individual and devices to encounter the physical world meets artificially created virtual spaces constructing the "Internet of Things (IoT) nirvana". In "Internet of Things (IoT)", "Things" refers to every object

that can establish connection to the Internet from anywhere and at any time, such objects are denoted as smart object [1]. As we know, everything in this world has good as well as bad characteristics. Due to the increase in the use of internet, the chances of attacks in IoT devices also have increased. “Symantec” disclosed the first IoT malware in November 2013, demonstrating the significance of having solutions to mitigate this security threat [1].

Malware is a generic phrase for all kinds of venomous software whose objective is to hack a computer system to violate the security guidelines in terms of availabilty, confidentiality or integrity [2]. The best-known forms of malware, viruses and worms, are characterised more by how they spread than by any particular features. The analysis of IoT malware and the discovery of software vulnerabilities are important components of modern security research [3].

Our Contribution:

- Designing and implementing a benchmark IoT-23 datasets, where 20 num-bers of files are considered as a combination of malicious and non malicious record specifically tailored for IoT devices, encompassing real-world network traffic with both benign and malicious activities. This dataset takes into account the diversity of IoT device types, communication protocols, and malware families, ensuring a comprehensive evaluation.

- Conducting a comparative analysis of various deep learning techniques, including CNN, RNN, autoencoder, MLP, RBN. This analysis allows for a comprehensive evaluation of the effectiveness of each technique in detecting malware on IoT devices.

- Evaluating the performance of the deep learning models based on key metrics such as accuracy. This assessment provides a quantitative measure of the models’ effectiveness in detecting malware.

- Providing insights into the strengths and weaknesses of different deep learn-ing techniques, identifying the most promising approaches for malware detection on IoT devices. These findings contribute to the development of more robust and efficient malware detection systems for IoT devices, enhancing their security and mitigating the risks associated with malware attacks.

The rest of the paper is organised as follows: Section2describes the back-ground and related work to resist malware attacks using machine learning and deep learning. Section3describes malware attacks IoT devices includes different deep learning techniques to resist malware attacks on IoT devices. Section4describes methodology. Section5presents the experimental results and analysis with

comparison of previous work and finally a conclusion has been formulated in Section6.

2 Literature review of the malware detectionapproaches

Researchers used various types of techniques to resist the increasing number of malware attacks. In paper [4], author proposed a multi-modal deep-learning techniques to detect android malware using various features. Authors use many static features to express the characteristics of different apps. Authors retrieved features from seven distinct types of files and utilized them in their work. Furthermore, authors recommend an efficient feature vector generation technique, which is suitable for detecting malware that resembles legitimate apps.

In paper [5], based on similarities in their behavior, authors proposed a novel intelligent malware analysis framework has been created for the dynamic and static analysis of malware samples. According to the author, the J48 Deci-sion tree performs the best in terms of accuracy and precision. The authors used 220 samples of files for analysis. This could be biased because not all features may have been taken into account with these samples.

In paper [6], authors used recurrent neural network deep learning tech-niques to construct a GRU architecture to identify malware in the Android operating system. The comparison between conventional machine learning methods and deep learning methodologies is presented to help author select the model that is most effective in detecting Android malware. The classifiers proposed by the authors were trained on a dataset taken from the CICAnd-Mal2017 dataset; it should be noted that the dataset was examined through a static study of actual, real-world examples of malicious and non-malicious Android applications. The authors addressed both API calls and permissions, as the use of both suggested that it was worth relying more on Android malware model identification.

In paper [7], authors proposed DLGraph, based on graph embedding and deep learning, is a novel method for malware detection. The proposed deep learning architecture (SDAs) consists of two stacked denoising autoencoders. The function call graphs of programmes can be learned from a single SDA as
a latent representation. The other SDA can hold a latent representation of the Windows API calls of programmes. The authors used the node2vec method to embed a function call graph in a feature space.

In paper [8], the author proposed a deep learning techniques to characterize malware and merge information from static and dynamic analysis of Android apps. The author has implemented DroidDetector, an online malware detection tool for Android that uses Deep Learning to determine whether an app is malicious or not.

In paper [9], the author mentioned about the behavior of security threats to the cyber-physical system is profiled using IP reputation systems. The poor performance of current reputation systems is due to their high administrative costs, false positive rate, and long usage times. Earlier open systems used a very small number of data sources to estimate the reputation of IP addresses. The authors proposed a novel hybrid strategy based on dynamic malware analysis, cyber threat intelligence, machine learning (ML), and data forensics to solve the above problem. IP reputation is anticipated by author using Big Data forensics in the pre-acceptance phase, and the associated zero-day attacks are characterised by behavioural analysis using the Decision Tree technique.

In paper [10], using static analysis and recent developments in image-based deep learning classification, the author has proposed a novel method for detect-ing Java bytecode malware. Jadeite, implemented by the author, extracts the Interprocedural Control Flow Graph (ICFG) from a given Java bytecode file, cleans it up, and then converts it into an adjacency matrix. Based on this matrix, Jadeite finally creates a grayscale image. To determine maliciousness, the author uses an object recognition technique in a Deep Convolutional Neu-ral Network (CNN) classifier. In addition, Jadeite pulls another set of features from the Java malware programme to improve malware classification. The retrieved images and these features are combined, and the CNN classifier uses them as inputs.

Dutta et al. introduced an ensemble technique using a meta-classifier (i.e., logistic regression) based on the principle of batch generalisation together with deep models such as Deep Neural Network (DNN) and Long Short-Term Mem-ory (LSTM). A Deep Sparse AutoEncoder (DSAE) is used for the feature engineering challenge in the first step of data preprocessing. In the second stage, a stacking ensemble learning strategy is used for classification. The effec-tiveness of the method described in this study is evaluated on a number of datasets, including Internet of Things (IoT) data (IoT-23, LITNET-2020, and NetML-2020) [11].

Abdalgawad et al. showed that generative deep learning techniques such as Adversarial Autoencoders (AAE) and Bidirectional Generative Adversarial Networks (BiGAN) can be used to detect intrusions by analysing network data.The results of this research show that the models based on GAN are superior in classifying and detecting attacks. In addition, an attempt was made to randomise the test set to introduce new data, and the model was able to detect it as an anomaly[12].

Banaamah and Ahmad [13] presented a study on the application of deep learning techniques to detect intrusions into IoT devices. The author used a common dataset called Bot-IoT for IoT intrusion detection. For IoT intrusion

detection, the author also used a set of deep learning techniques including Convolutional Neural Network, Gated Recurrent Unit, and Long Short Mem-ory Neural Network. The author evaluated the proposed model and compared it with current methods. The results of the experiments showed the potential usefulness of the proposed approach for intrusion detection.

To detect malware in IoT devices, Riaz et al. [14] proposed a Deep Learning-based ensemble classification method called CNN-CNN. A three-step process is used: first, data is preprocessed by scaling, normalization, and noise reduction; second, features are selected; third, hot coding is used; and finally, an ensemble classifier based on CNN and LSTM outputs is used to detect malware. The proposed method has not been validated in a real-time environment, and other limitations of this research include considering only static malware detection, using numerous instances of a single dataset, and excluding CPU and RAM.

3 Malware attacks on IoT Devices

IoT devices are more vulnerable to various threats due to their low power consumption and limited processing power . Occasionally, malware can be installed by hackers on edge devices, causing these devices to send faulty or altered information to the cloud server via enterprise information systems. In IIoT net-works, this type of malware attack usually results in financial and reputational damage [15]. IoT malware has several characteristics, including using DDoS attacks, scanning open ports for IoT services such as FTP, SSH, or Telnet, and performing a brute force attack to gain access to IoT devices. According to author, most of today's malware was created by copying the source code according to instructions found on the Internet or by using a different version of the same malicious code by the creator of the malware. Smart meters, med-ical devices, public safety sensors, and many other connected devices make up the IoT. Many IoT malware families, including Aidra, Bashlite, and Mirai, can use scanners to find unprotected ports and default credentials on these devices [16]. By starting a short scan phase where TCP SYN sends probes to random IPv4 addresses on Telnet TCP ports 2323 and 23, infected IoT devices can be identified. When an IoT device responds to the probe, the attack moves to the brute force login phase. In this step, an attacker attempts to establish a Telnet connection by using a username and password combination from a pre-built credentials table. The victim's IP address and the credentials used are forwarded to a collection server if Telnet access is allowed. The tool monitors a command-and-control server that indicates the intended victim of an attack [17]. Another malware that is notorious for causing problems in IoT devices is called Hajime. Hajime works similarly to Mirai by spreading through unpro-tected open Telnet ports and using the same username and password tables. Unlike Mirai, however, Hajime is connected via a peer-to-peer network. The message will eventually propagate to all

other peers after the controller has submitted policies to its peer network. This makes it harder to take it out with a strong design.

The creation of a security architecture for IoT-based enterprise information systems is therefore always the focus of research. Most conventional methods for detecting malware in enterprise information systems use feature extraction approaches, where relevant features are extracted from the code and analysed to find the infection [15]. However, due to the limited power supply and pro-cessing capacity of IoT devices, these feature identification approaches do not work effectively on IoT devices in enterprise information systems. In addi-tion, adversarial attacks, in which attackers alter training patterns so that the malware detection system cannot train properly, render traditional malware detection systems useless. In the IoT environment of enterprise information systems, static, dynamic, and hybrid methods are used for malware detec-tion. Static analysis includes n-grams, OpCode, and signature-based detection, while dynamic detection involves running an application in a virtual environ-ment. For malware detection in IoT environment, researchers have presented various machine learning approaches such as RNN, LSTM, etc. Currently, cryptographically linked transitions are recorded and verified using blockchain technology as the ledger system [17]. Blockchain can bring new potential to the Internet of Things in many ways. Its high efficiency, data security, credi-bility, and low cost have helped it gain wide appeal in the technology sector. Any kind of virus can be detected using blockchain, and it can also be used to protect one's infrastructure from potential threats.The secret to protecting these huge application areas is machine learning. A learning machine offers the best chance for detection, as traditional malware detection software is unable to meaningfully keep up with malware growth.

3.1 Deep Learning Techniques

Deep learning is a type of machine learning that uses many layers to progres-sively extract higher-order features from raw input. These neural networks strive to mimic how the human brain works, but are far from capable of "learning" from massive amounts of data. Additional hidden layers can help optimise and refine accuracy, even though a neural network with only one layer can still make approximate predictions.

3.1.1 Convolutional Neural Networks

CNNs, often referred to as ConvNets, consist of multiple layers and are mostly used for object recognition and image processing. When it was still known as LeNet, Yann LeCun created the first CNN in 1988. It was used to decode characters such as ZIP codes and numbers.A convolutional layer, a pooling layer,

and a fully connected layer (FC) are the three layers that make up a deep learning CNN. The first layer is the convolutional layer, while the last layer is the FC layer.

Let's assume we have a CSV dataset with N samples and D features. Each sample can be represented as a row in the dataset, and each feature can be represented as a column. Input Layer: The input data from the CSV dataset can be represented as X of shape N × D.

Reshape Layer: Since we are dealing with a non-image dataset, we need to reshape the input data into a 3D array to simulate imagelike structures. We can reshape X to obtain X_reshaped of shape N × W × H, where W represents the width and H represents the height of the imagelike structure.
Convolutional layer: The convolutional layer, the central component of a CNN, is where most of the computation takes place. The first convolutional layer may be followed by another convolutional layer. A kernel or filter within this layer moves over the receptive fields of the image during the convolution process to determine if a feature is present.In the convolutional layer, we apply filters to the input data to extract features. The output feature maps after convolution can be represented as:

$$X\_conv = f\_conv(X\_reshaped, W\_conv) + b\_conv,$$

where f_conv represents the convolution operation, W_conv represents the convolutional filters (weights), and b_conv represents the biases.

Activation Function: An activation function is applied element-wise to the output of the convolutional layer to introduce non-linearity.The output after applying the activation function can be represented as:

$$X\_activation = f\_activation(X\_conv),$$

where f_activation is the chosen activation function.

Pooling layer: The pooling layer sweeps over the input image with a kernel or filter, similar to the convolutional layer. Unlike the convolution layer, the pooling layer has fewer input parameters, but some information is also lost. On the positive side, this layer simplifies the CNN and increases its effectiveness. Pooling layers down sample the feature maps, reducing their spatial dimensions while preserving important information. The output after pooling can be represented as:

$$X\_pool = f\_pool(X\_activation),$$

where f_pool represents the pooling operation.

Flatten Layer: The pooled feature maps are flattened into a 1D vector to prepare them for fully connected layers.The flattened output can be represented as:

$$X\_flattened = flatten(X\_pool),$$

where flatten is the operation that converts a 2D array to a 1D vector.
Fully Connected layer: Based on the features extracted in the previous layers, image categorization in the CNN takes place in the FC layer. Fully connected in this context means that each activation unit or node of the subsequent layer is connected to each input or node of the previous layer.The flattened feature vector is passed through fully connected layers.The output after the fully connected layers can be represented as:

$$X\_fc = f\_fc(X\_flattened, W\_fc) + b\_fc,$$

where f_fc represents the fully connected operation, W_fc represents the fully connected weights, and b_fc represents the biases.

Output Layer: The output layer typically applies a suitable activation function depending on the task. The final output of the CNN can be represented as:

$$y = f\_output(X\_fc),$$

where f_output is the activation function applied to the output of the fully connected layer.

3.1.2 Recurrent Neural Networks

An artificial neural network that uses sequential data or time series data is called a recurrent neural network (RNN). These deep-learning algorithms are incorporated into popular programs such as Siri, Voice Search, and Google Translate, and are commonly used for ordinal or temporal queries in language translation, natural language processing (NLP), speech recognition, and image captioning. Recurrent neural networks (RNNs) use training data to learn, just like feedforward and convolutional neural networks (CNNs). They are characterized by their "memory," which allows them to influence current input and output by using data from previous inputs. The outputs of recurrent neural networks depend on the previous parts of the sequence, unlike typical deep neural networks that assume that inputs and outputs are independent. Unidirectional recurrent neural networks

are not able to consider future events in their predictions, although they would be useful for deciding the output of a particular sequence.

Suppose a dataset with N samples and D features. Each sample can be represented as a row in the dataset, and each feature can be represented as a column.

Input Layer:The input data from the CSV dataset can be represented as X of shape N × D.

Recurrent Layer:In the recurrent layer, we process the sequential data by considering the temporal dependencies between the samples. The output of the recurrent layer at each time step can be represented as:

$$H\_t\_ = f\_rnn(X\_t\ , H\_t1, W\_rnn) + b\_rnn$$

where f_rnn represents the recurrent operation, X_t represents the input at time step t, H_t represents the hidden state from the previous time step, W_rnn represents the recurrent weights, and b_rnn represents the biases.

Output Layer:The output layer transforms the hidden state of the recurrent layer into the desired output. It can be represented as:

$$y\_t = f\_output(H\_t, W\_output) + b\_output,$$

where f_output represents the output operation, H_t represents the hidden state at time step t, W_output represents the output weights, and b_output represents the biases.

Time Unrolling:To process the entire sequence, we perform the recurrent layer and output layer operations for each time step.

For t = 1 to T (where T is the sequence length):

$$H\_t = f\_rnn(X\_t, H\_t1, W\_rnn,$$

$$y\_t = f\_output(H\_t, W\_output) + b\_output$$

Final Output:The final output of the RNN can be represented as the sequence of output values:

$$y = [y\_1, y\_2, \ldots, y\_T].$$

3.1.3 Autoencoder

The output layer of a neural network with autoencoder has the same dimensionality as the input layer. In other words, there are exactly as many output units in the output layer as input units in the input layer. An autoencoder, also called a replicator neural network, duplicates data from input to output in an unsupervised manner.

An autoencoder consists of three parts:

Encoder:A fully interconnected feedforward neural network, called an encoder, encodes an input image into a compressed representation in a smaller dimension after compressing the input into a latent spatial representation. The original image has been distorted in the compressed form. The output of the encoder can be represented as:

$$z = f\_encoder(x),$$

where x represents the input data and f_encoder is the encoder function.

Decoder:The decoder is also a feedforward network with a topology similar to that of the encoder. This network has the task of translating the input from the code back to its original dimension.The output of the decoder can be represented as:

$$\hat{x} = f\_decoder(z),$$

where $\hat{x}$ represents the reconstructed input data and f_decoder is the decoder function.

Loss Function: The loss function measures the discrepancy between the original input data and the reconstructed output data. It quantifies the reconstruction error and provides a training signal for the autoencoder to learn meaningful representations.

The loss function can be represented as:

$$L(x, \hat{x}),$$

where L is a suitable loss function, such as mean squared error (MSE), binary cross-entropy, or Kullback-Leibler divergence.

Training: During training, the autoencoder aims to minimize the loss function by adjusting its parameters (encoder and decoder weights) using techniques like gradient descent. The training objective can be represented as:

$$minimizeL(x, \hat{x}),$$

where the objective is to minimize the discrepancy between the original input data and the reconstructed output data.

3.1.4 Multi Layer Perceptron

A neural network with a multi-layer perceptron has a nonlinear mapping between inputs and outputs A multilayer perceptron consists of an input layer, an output layer, and one or more hidden layers, each consisting of multiple neurons arranged one above the other. The neurons in a multilayer perceptron can use any activation function, unlike the neurons in a perceptron, which must have an activation function that enforces a threshold, such as ReLU or sigmoid. Similar to the human brain, a multilayer perceptron consists of interconnected neurons that communicate with each other. Each neuron is assigned a value. The network consists of three basic layers:

Let's consider an MLP with L layers (including the input and output layers) and denote the number of neurons in each layer as follows:

Input Layer:This is the lowest layer of the network, where an input is made to produce an output. Input Layer: N_input neurons

Hidden Layer(s): There must be at least one hidden layer in the network. To produce something useful, the hidden layer(s) perform computations and operations on the input data.

Hidden Layers: N_1, N_2, ..., N_(L−2) neurons

For the first hidden layer:

$$a\_1 = f(W\_1 * x + b\_1),$$

where a_1 represents the activation values of the neurons in the first hidden layer, f is the activation function (applied elementwise), W_1 represents the weight matrix for the connections between the input layer and the first hidden layer, x represents the input data, and b_1 represents the bias vector for the first hidden layer.

For the subsequent hidden layers (l = 2 to L−2):

$$a\_l = f(W\_l * a\_(l-1) + b\_l),$$

where a_l represents the activation values of the neurons in the l-th hidden layer, W_l represents the weight matrix for the connections between the (l−1)th hidden layer and the lth hidden layer, and b_l represents the bias vector for the lth hidden layer.

Output Layer:The neurons of this layer produce information that is meaningful.

Output Layer: N_output neurons

For the output layer:

$$y = f(W\_out * a\_(L-2) + b\_out),$$

where y represents the output values of the MLP, W_out represents the weight matrix for the connections between the last hidden layer and the output layer, and b_out represents the bias vector for the output layer.

- **Activation Function**: An activation function f is applied element-wise to the weighted sum of inputs for each neuron in the MLP. Common activation functions include sigmoid, tanh, ReLU, or softmax (for multi-class classification).
- **Loss Function:** The loss function measures the discrepancy between the MLP's predicted output and the desired output. The choice of the loss function depends on the specific problem, such as mean squared error (MSE) for regression or cross-entropy for classification.
- **Backpropagation and Weight Updates:** The backpropagation algorithm calculates the gradients of the loss function with respect to the weights and biases of the MLP, allowing for weight updates to minimize the loss through techniques like gradient descent or its variants.

The weight update process can be represented as:

$$W\_new = W - \eta * \frac{\partial L}{\partial W},$$

$$b\_new = b - \eta * \frac{\partial L}{\partial b},$$

where W_new and b_new represent the updated weights and biases, η is the learning rate, and $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial b}$ represent the gradients of the loss function with respect to the weights and biases, respectively.

3.1.5 Radial Basis Function Networks

A special type of artificial neural networks used for function approximation questions are radial basis function (RBF) networks. RBF networks are three-layered, use a universal approximation, and learn faster than other neural networks. A neural network with radial basis functions usually consists of three layers:

Input layer: Each predictor variable has a single neuron in the input layer. Each neuron in the hidden layer receives the value from the input neurons. Categorical values are represented by N-1 neurons, where N is the total number of categories. By removing the median from the equation and dividing by the inter-quartile range, the range of values is standardized.

The input data can be represented as x, a vector of input features.

Hidden layer: The number of neurons in the buried layer varies. (The ideal number is determined by the training process). A point-centered radial basis function is part of each neuron. The number of predictor variables and the number of dimensions coincide. For each dimension, the radius or span of the RBF function may change.

For each neuron in the hidden layer i (with M neurons in total), the output can be represented as:

$$h\_i = \phi(\| x - c\_i \|),$$

where h_i represents the output of the ith hidden neuron, $\| x - c\_i \|$ represents the Euclidean distance between the input x and the center c_i of the i-th neuron, and ϕ is a radial basis function that determines the activation level based on the distance.

Summation layer:A weight assigned to the neuron is multiplied by the value obtained from the hidden layer before being passed to summation. Here, the weighted values are summed, and the result is displayed as the output of the network. Each target category in a classification problem has a single output, where the value represents the probability that the evaluated case belongs to that category.

The output y can be represented as a weighted sum of the hidden layer outputs:

$$y = \sum(w\_i * h\_i) + b,$$

where w_i represents the weights associated with each hidden neuron, b represents the bias term, and the summation is performed over all M hidden neurons.

Training: The training of an RBFN typically involves two steps: center selection and weight adjustment.

Center Selection: The centers of the radial basis functions can be selected using techniques like k-means clustering or random sampling from the input data.

Weight Adjustment: The weights and bias terms are adjusted using techniques like least squares or gradient descent to minimize the discrepancy between the network's output and the desired output.

4 Proposed Data Preprocessing

In the implementation of this research, an idea can be depicted from figure1, as a deep learning architectural model used to build a comprehensive IoT security model that increases accuracy in identifying security concerns. Figure 2displays the data pre-processing model in details. The preprocessing of thedatasets is depicted in the first section of the picture. With the help of CNN, RNN, Autoencoder, MLP, and RBFN, the subsequent classification phase was carried out. After that, our model was trained, put to the test, and assessed.

4.1 Dataset IoT-23

IoT-23 is a dataset of network traffic from Internet of Things (IoT) devices. In IoT devices, it captured 20 malware executions and 3 benign IoT device traffic captures. With images from 2018 to 2019, it was first published in January 2020. The Stratosphere Lab, AIC Group, FEL, CTU University, Czech Republic, is where this Internet of Things network traffic was recorded.Conn.log.labeled files generated by the execution of the network analyzer Zeek, as well as various characteristics and information about each of the records are included in the record in its entire form. pcap files, which are the original network capture files, are also included.There are 325,307,990 observations altogether in the collection, of which 294,449,255 are malicious.
Let:
N be the total number of records in the dataset (N = 1,444,674 in this case). x be a record in the dataset, where each record consists of multiple features and is represented as a vector.

4.2 Data Preprocessing

4.2.1 Data Cleaning and Preparation:

- Drop irrelevant columns (e.g., "tunnel parents").
- Load each record from the IoT-23 dataset into separate data frames, skipping the first 10 rows and reading the following 100,000 rows.
- Concatenate the 23 data frames into a single new data frame.
- Convert the data frame to a CSV file, resolving compatibility issues.
- Drop the extra "Unnamed" column generated during conversion.
- Convert string data to integers.
- Use statistical correlation to filter out data not belonging to the "label" column.
- Drop variables that have no impact on the results.

4.2.2 Creation of Combined Dataset:

- Create the file "iot_23_combined.csv" containing the merged data set.
  - The combined dataset contains a total of 1,444,674 records.

4.2.3 Conversion of String Data and Handling Missing Data:

All string data in the dataset is converted into integers.
Missing data is filled in during the conversion process.

Let

x_converted = ConvertStringToInt(x_cleaned)

The function ConvertStringToInt converts the string data in x cleaned to integer format and fills in missing data.

4.2.4 Elimination of Columns:

- Based on Figure3, the columns 'local_orig' and 'local_resp' are eliminated from the dataset.
- These columns are removed due to a large amount of missing data, and no correlations were found between them using correlation matrices.

Let
x_cleaned = x without the 'local_orig' and 'local_resp' columns.

4.2.5 Data Normalization:

- The final step is the normalization of the data.
- Normalization involves transforming the data to a common scale.
- Neural networks typically require input data to be normalized to avoid issues with negative values.
- Normalize the data between 0 and 1 to avoid negative values, making it suitable for neural networks.

Let x_normalized = Normalize(x_final)
The function Normalize transforms each feature in x final to a normalized value between 0 and 1.

4.2.6 Feature Selection and Classification:

- Select the best features for the model. The selection of appropriate features is crucial for the model's performance. 19 features were chosen for the IoT-23 dataset.
- Utilize various neural network models, including CNN, RNN, autoencoder, MLP, and RBFN, to predict attacks.
- Python, TensorFlow, and Keras are used to implement these models.

Let

x_selected = SelectFeatures(x_normalized)

The function SelectFeatures chooses the 19 best features from x_normalized based on some criteria.

Classification:

CNN: CNN(x_selected)
RNN: RNN(x_selected)
Autoencoder: Autoencoder(x_selected)
MLP: MLP(x_selected)
RBFN: RBFN(x_selected)

Each function represents the implementation of the respective neural network model using the selected features.

4.2.7 Training and Testing:

- Splitting the dataset into training and testing sets: Let: x_train, x_test = SplitDataset(x_selected).
- The function SplitDataset splits the dataset x_selected into training and testing sets, with a ratio of 80% for training and 20% for testing.
- Training the models: TrainModel(model, x_train)
- The function TrainModel trains a given model using the training set x_train.

Testing the models: TestModel(model, x_test)

The function TestModel evaluates the performance of a trained model using the testing set x_test.

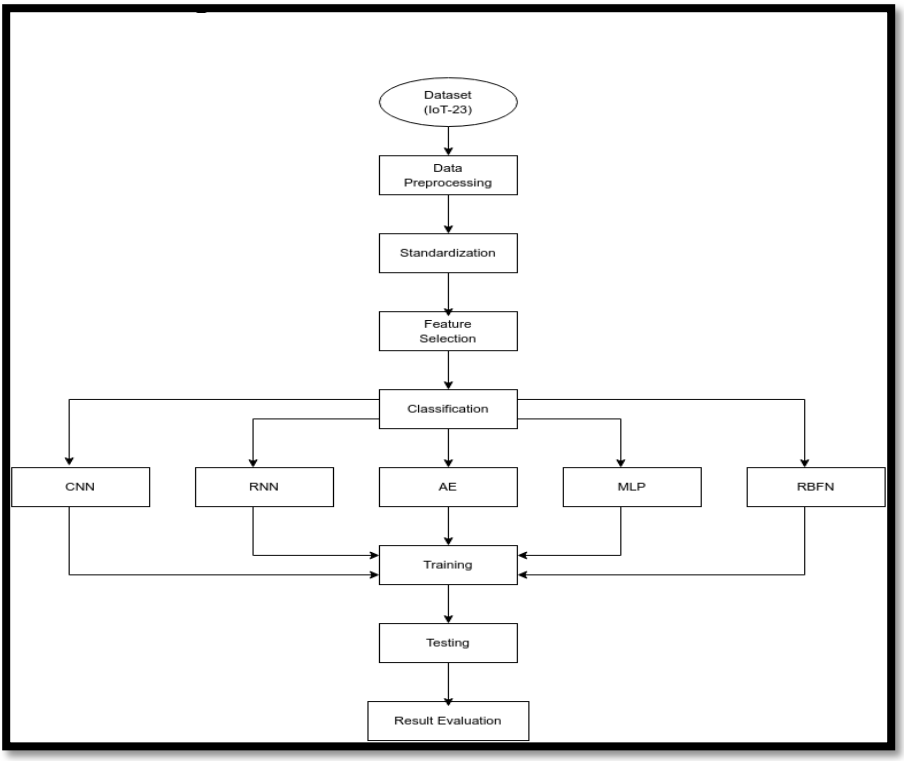

Fig. 1 Architectural Model

5 Result and Analysis

The experiment results are shown in Table1. The results of each method are compared, taking into account the accuracy and the time required to perform each algorithm. Table2shows the results using the data preprocessing tech-nique used in paper [18]. The preprocessing model method in this paper shows more accuracy rate for CNN, MLP and RBFN models 0.995,0.99,0.98 respectively whereas RNN shows same accuracy rate as mentioned in paper [18] and the accuracy rate in Autoencoder method is differ by 0.042. The cost of time for CNN method is better as the accuracy rate is also higher. Figure4shows a data comparison result of proposed work along with paper [18].

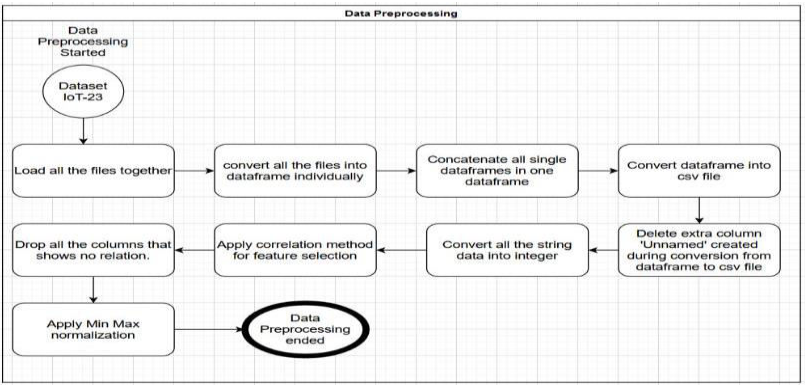

Fig. 2 Data preprocessing

Table 1 Experiment Results

Methods Implement	Accuracy	Time Cost (s)
CNN	0.995	7166.996
RNN	0.916	2134.948
Autoencoder(AE)	0.925	393.933
MLP	0.99	898.545
RBFN	0.98	244.314

Table 2 Experiment Results using data preprocessing method mentioned in paper [18]

Methods Implement	Accuracy	Time Cost (s)
CNN	0.692	7396.231
RNN	0.916	6849.035
Autoencoder(AE)	0.967	346.557
MLP	0.693	705.264
RBFN	0.693	209.286

6 Conclusions

In conclusion, our comparative analysis of different deep learning techniques for malware detection on IoT devices indicates that CNN (Convolutional Neu-ral Network), RBFN (Radial Basis Function Network), and MLP (Multi-Layer Perceptron) demonstrate better results compared to Autoencoder and RNN(Recurrent Neural Network).CNN excels in capturing spatial dependencies and extracting relevant features from IoT network traffic data, making it highly effective in detecting malware patterns. RBFN and MLP also show promising results, indicating their suitability for malware detection on IoT devices.On the other hand, Autoencoder and RNN yield comparable results, suggesting that these techniques may have limitations when applied to the specific challenges posed by IoT device environments.It is important to consider the unique characteristics of the IoT devices, such as limited resources, diverse communication protocols, and dynamic environments, when selecting the appropriate deep learning technique. Additionally, factors like computational efficiency and accuracy trade-offs should be taken into account. The findings of this research contribute to the advancement of malware detection systems for IoT devices, helping to enhance their security and mitigate the risks associated with malware attacks. Future research can focus on refining and optimizing the identified techniques, as well as exploring novel approaches that leverage the strengths of CNN, RBFN, and MLP in IoT malware detection.

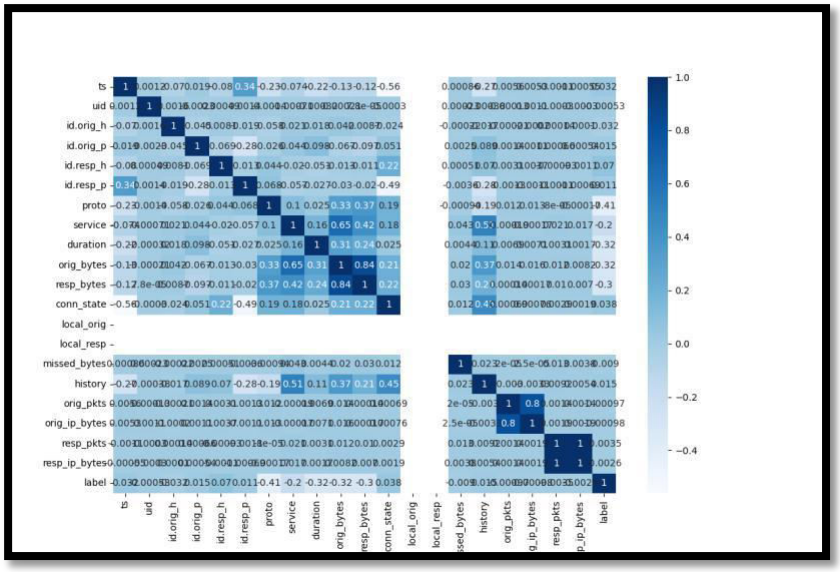

Fig. 3 Correlation between all the files

Table 3 Results Comparison with Paper[18]

Methods Implement	Accuracy(Own)	Accuracy (Paper[[18]])
CNN	0.995	0.692
RNN	0.916	0.916
Autoencoder(AE)	0.925	0.967
MLP	0.99	0.693
RBFN	0.98	0.693

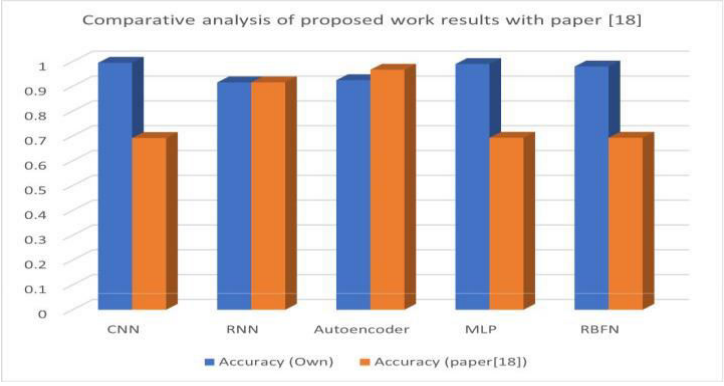

Fig. 4 Comparative analysis of proposed work results with paper [18]

References

[1] Buyya, R., Dastjerdi, A.V. (eds.): Internet of Things: Principles and Paradigms. Elsevier, 50 Hampshire Street, 5th Floor, Cambridge, MA 02139, USA (2016)

[2] Creutzburg, R. (ed.): Handbook of Malware. Technische Hochschule Brandenburg, Germany (2016)

[3] Zhang, Z.K., Cho, M.C.Y., Wang, C.W. C.W.and Hsu, Chen, C.K., Shieh, S.: IoT security: ongoing challenges and research opportunities. In 2014 IEEE 7th international conference on service-oriented computing and applications 230–234. November 2014 (2014)

[4] Kim, T., Kang, M. B.and Rho, Sezer, S., Im, E.G.: A multimodal deep learning method for android malware detection using various features. IEEE Transactions on Information Forensics and Security 14(3), 773–778 (2018)

[5] Sethi, K., Chaudhary, S.K., Tripathy, B.K., Bera, P.: A novel malware analysis framework for malware detection and classification using machine learning

approach. In Proceedings of the 19th international conference on distributed computing and networking, 1–4 January 2018 (2018)

[6] Sabhadiya, J. S.and Barad, Gheewala, J.: Android malware detection using deep learning. In 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), IEEE 1254–1260 April 2019 (2019)

[7] Jiang, T. H.and Turki, Wang, J.T.: DLGraph: Malware detection using deep learning and graph embedding. In 2018 17th IEEE international conference on machine learning and applications (ICMLA), 1029–1033 December 2018 (2018)

[8] Yuan, Z., Lu, Y., Xue, Y.: Droiddetector: android malware characteriza-tion and detection using deep learning. Tsinghua Science and Technology 21(1), 114–123 (2016)

[9] Usman, N., Usman, F. S.and Khan, Jan, A. M.A.and Sajid, Alazab, M., Watters, P.: Intelligent dynamic malware detection using machine learning in ip reputation for forensics data analytics. Future Generation Computer Systems 118, 124–141 (2021)

[10] Obaidat, I., Sridhar, K.M. M.and Pham, Phung, P.H.: Jadeite: A novel image-behavior-based approach for java malware detection using deep learning. Computers & Security 113, 102547 (2022)

[11] Dutta, V., Chora´s, M., Pawlicki, M., Kozik, R.: A deep learning ensemble for network anomaly and cyber-attack detection. Sensors 20(16), 4583 (2020).

[12] Abdalgawad, A. N.andSajun, Kaddoura, Y., Zualkernan, I.A., Aloul, F.: Generative deep learning to detect cyberattacks for the IoT-23 dataset. IEEE Access 10, 6430–6441 (2021).

[13] Banaamah, A.M., Ahmad, I.: Intrusion Detection in IoT Using Deep Learning. Sensors 22, 8417 (2022).

[14] Riaz, S., Latif, S., Usman, S.M., Ullah, S.S., Algarni, A.D., Yasin, A., Anwar, A., Elmannai, H., Hussain, S.: Malware Detection in Internet of Things (IoT) Devices Using Deep Learning. Sensors 22, 9305 (2022).

[15] Gaurav, A., Gupta, B.B., Panigrahi, P.K.: A comprehensive survey on machine learning approaches for malware detection in iot-based enter-prise information system. Enterprise Information Systems 17(3), 439–463 (2023)

[16] Ngo, Q.D., Nguyen, V.H. H.T.and Le, Nguyen, D.H.: A survey of iot malware and detection methods based on static features. ICT Express 6(4), 280–286 (2020)

[17] Clincy, V., Shahriar, H.: IoT malware analysis. In 2019 IEEE 43rd annual computer software and applications conference (COMPSAC) 920–921. July 2019 (2019)

[18] Liang, Y., Vankayalapati, N.: Machine Learning and Deep Learning Methods for Better Anomaly Detection in IoT-23 Dataset Cybersecu-rity. Preprint. Available online: https://github. com/yliang725/Anomaly-Detection-IoT23 (2022)