Hindawi
Security and Communication Networks
Volume 2020, Article ID 8890306, 11 pages
https://doi.org/10.1155/2020/8890306

Research Article

DL-IDS: Extracting Features Using CNN-LSTM Hybrid Network for Intrusion Detection System

Pengfei Sun,[1] Pengju Liu,[2] Qi Li,[3] Chenxi Liu,[2] Xiangling Lu,[2] Ruochen Hao,[2] and Jinpeng Chen [1]

[1]*School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, Beijing 100876, China*
[2]*School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China*
[3]*Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China*

Correspondence should be addressed to Jinpeng Chen; chenjinpeng@nlsde.buaa.edu.cn

Received 6 April 2020; Revised 17 July 2020; Accepted 25 July 2020; Published 28 August 2020

Academic Editor: Huaming Wu

Many studies utilized machine learning schemes to improve network intrusion detection systems recently. Most of the research is based on manually extracted features, but this approach not only requires a lot of labor costs but also loses a lot of information in the original data, resulting in low judgment accuracy and cannot be deployed in actual situations. This paper develops a DL-IDS (deep learning-based intrusion detection system), which uses the hybrid network of Convolutional Neural Network (CNN) and Long Short-Term Memory Network (LSTM) to extract the spatial and temporal features of network traffic data and to provide a better intrusion detection system. To reduce the influence of an unbalanced number of samples of different attack types in model training samples on model performance, DL-IDS used a category weight optimization method to improve the robustness. Finally, DL-IDS is tested on CICIDS2017, a reliable intrusion detection dataset that covers all the common, updated intrusions and cyberattacks. In the multiclassification test, DL-IDS reached 98.67% in overall accuracy, and the accuracy of each attack type was above 99.50%.

1. Introduction

1.1. Background. In recent years, with the rapid development of emerging communications and information technologies such as 5G communications, mobile Internet, Internet of Things, "cloud computing," and big data, network security has become increasingly important. As an important research content of network security, intrusion detection has been paid attention by experts and scholars. Problems that are common under traditional anomaly-based detection methods include the inaccurate feature extraction of network traffic and difficulty in building attack detection models, which leads to high false alarm rate when judging attack traffic. It is difficult for network security personnel to find unknown threats, which makes the defense inherently passive. In other words, traditional methods are no longer applicable to today's Internet as per its massive data scale.

In recent years, many scholars have explored how to use artificial intelligence (AI) to detect and analyze network traffic for intrusion detection and defense systems. Hassan et al. [1] proposed an ensemble-learning model based on the combination of a random subspace (RS) learning method with random tree (RT), which detected cyberattacks of SCADA by using the network traffic from the SCADA-based IoT platform. Khan and Gumaei [2] compared the most popular machine learning methods for intrusion detection in terms of accuracy, precision, recall, and training time cost. Alqahtani et al. [3] proposed GXGBoost model to detect intrusion attacks based on a genetic algorithm and an extreme gradient boosting (XGBoost) classifier. Derhab et al. [4] proposed a security architecture that integrates the Blockchain and the software-defined network (SDN) technologies, which focuses on the security of commands in industrial IoT against forged commands and misrouting of

commands. The current mainstream methods are intrusion detection systems based on machine learning (ML) or deep learning (DL). Among them, the ML-based system mainly classifies and detects network traffic by analyzing the manually extracted features of network traffic, while the DL-based system can not only analyze the manually extracted features but also automatically extract the features from the original traffic. Therefore, DL-based systems can circumvent the manual feature extraction problem and enhance the detection accuracy compared to general ML-based systems.

To achieve higher accuracy, DL-based intrusion detection methods require a large amount of data for training, especially different types of attack traffic data. In the actual environment and the existing datasets [KDD99, NSL-KDD, and CICIDS2017], the attack traffic is always less compared with normal traffic. Moreover, because some types of attack traffic are difficult to capture and simulate, the amount of data available for model training is particularly small. These problems greatly restrict the accuracy of the DL-based method, making it difficult to judge certain types of attacks.

1.2. Key Contributions. This paper proposes a DL-based intrusion detection system, DL-IDS, which uses the hybrid network of Convolutional Neural Network (CNN) and Long Short-Term Memory Network (LSTM) to extract the temporal and spatial features of network traffic data to improve the accuracy of intrusion detection. In the model training phase, DL-IDS uses category weights to optimize the model. This method reduces the effect of the number of unbalanced samples of several attack types in model training samples on model performance and improves the robustness of training and prediction. Finally, we test DL-IDS to classify multiple types of network traffic on the CICIDS2017 dataset and compare it with the CNN-only model, LSTM-only model, and other machine learning models because CICIDS2017 is a recent original network traffic dataset simulating real situations. The results show that DL-IDS reached 98.67% in overall accuracy, and the accuracy of each attack type was above 99.50%, which achieved the best results in all models.

1.3. Paper Organization. The remainder of this paper is organized as follows. Section 2 discusses the classification of abnormal traffic as per previously published studies. Section 3 describes in detail the datasets and data preprocessing methods we used in this study. The classifier structure and classification methods used for traffic classification under the proposed model are described in Section 4. Section 5 presents the results of our model evaluation with various hyperparameters. Section 6 summarizes the paper and discusses potential future development trends.

2. Related Work

With the continual expansion of the Internet, network security has become a problem that cannot be ignored. Malicious network behaviors such as DDos and brute force attacks tend to be "mixed into" malicious traffic. Security researchers seek to effectively analyze the malicious traffic in a given network so as to identify potential attacks and quickly stop them [5–8].

2.1. Traditional Intrusion Detection System. Traditional methods of intrusion detection mainly include statistical analysis methods [9], threshold analysis methods [10], and signature analysis methods [11]. These methods do reveal malicious traffic behavior; however, they require security researchers to input data related to their personal experience; to this effect, their various rules and set parameters are very inefficient. Said "experience" is also only a summary of the malicious traffic behavior found in the past and is typically difficult to quantify, so these methods cannot be readily adapted to the huge amount of network data and volatile network attacks of today's Internet.

2.2. Intrusion Detection System Based on ML. Advancements in machine learning have produced models that effectively classify and cluster traffic for the purposes of network security. Early researchers attempted simple machine learning algorithms for classification-clustering problems in other fields, such as the k-Nearest Neighbor (KNN) [12], support vector machine (SVM) [13], and self-organizing maps (SOM) [14], with good results on KDD99, NSL-KDD, DARPA, and other datasets. These datasets are out of date, unfortunately, and contain not only normal data but also attack data that are overly simple. It is difficult to use these datasets to simulate today's highly complex network environment. It is also difficult to achieve the expected effect using these algorithms to analyze malicious traffic in a relatively new dataset, as evidenced by our work in this study.

2.3. Intrusion Detection System Based on Deep Neural Network. The success of machine learning algorithms generally depends on data representation [15]. Representation learning, also called feature learning, is a technique in deep neural network, which can be used to learn the explanatory factors of variation behind the data. Ma et al. combine spectral clustering and deep neural network algorithms to detect intrusion behaviors [16]. Niyaz et al. used deep belief networks for developing an efficient and flexible intrusion detection system [17]. But these research methods construct their models to learn representations from manually designed traffic features, not taking full advantage of the ability of deep neural networks. Eesa et al. showed that higher detection rate and accuracy rate with lower false alarm rate can be obtained by using improved traffic feature set [18]. Learning features directly from traffic raw data should be feasible, such as in the fields of computer vision and natural language processing [19].

Two most widely used deep neural network models are CNN and RNN. The CNN uses original data as the direct input to the network, does not necessitate feature extraction or image reconstruction, has relatively few parameters, and requires relatively little data in process. CNNs have been proven to be highly effective in the field of image recognition

[20]. For certain network traffic of protocols, CNNs can perform well through fast training. Fan and Ling-zhi [21] extracted very accurate features by using a multilayer CNN, wherein the convolution layer connects to the sampling layer below; this model outperformed classical detection algorithms (e.g., SVM) on the KDD99 dataset. However, the CNN can only analyze a single input package—it cannot analyze timing information in a given traffic scenario. In reality, a single packet in an attack traffic scenario is normal data. When a large number of packets are sent at the same time or in a short period, this packet becomes malicious traffic. The CNN does not apply in this situation, which in practice may lead to a large number of missed alerts.

The recurrent neural network (RNN) is also often used to analyze sequential information. LSTM, a branch of RNNs, performs well in sequence information analysis applications such as natural language processing. Kim et al. [22] compared the LSTM-RNN network against Generalized Regression Neural Network (GRNN), Product-based Neural Network (PNN), *k*-Nearest Neighbor (KNN), SVM, Bayesian, and other algorithms on the KDD99 dataset to find that it was superior in every aspect they tested. The LSTM network alone, however, centers on a direct relationship between sequences rather than the analysis of a single packet, so it cannot readily replace the CNN in this regard.

Wu and Guo [23] proposed a hierarchical CNN + RNN neural network and carried out experiments on NSL-KDD and UNSW-NB15 datasets. Hsu et al. [24] and Ahsan and Nygard [25] used another CNN + LSTM model to perform multiclassification experiments on the NSL-KDD dataset. Hassan et al. [26] proposed a hybrid deep learning model to detect network intrusions based on CNN network and a weight-dropped, long short-term memory (WDLSTM) network. This paper mainly conducted experiments on UNSW-NB15 dataset. However, these studies are still based on extracted features in advance.

Abdulhammed et al. [27] used Autoencoder and Principle Component Analysis to reduce the CICIDS2017 dataset's feature dimensions. The resulting low-dimensional features from both techniques are then used to build various classifiers to detect malicious attacks. Musafer et al. [28] proposed a novel architecture of IDS based on advanced Sparse Autoencoder and Random Forest to classify the patterns of the normal packets from those of the network attacks and got good results.

In this study, we adopted a malicious traffic analysis method based on CNN and LSTM to extract and analyze network traffic information of network raw dataset from both spatial and temporal dimensions. We conducted training and testing based on the CICIDS2017 dataset that well simulates the real network environment. We ran a series of experiments to show that the proposed model facilitates very effective malicious flow analysis.

3. Datasets and Preprocessing

3.1. Dataset. The IDS is the most important defense tool against complex and large-scale network attacks, but the lack of available public dataset yet hinders its further development. Many researchers have used private data within a single company or conducted manual data collection to test IDS applications, which affects the credibility of their results to some extent. Public datasets such as KDD99 and NSL-KDD [29] are comprised of data encompassing manually selected stream characteristics rather than original network traffic. The timing of the data collection is also outdated compared to modern attack methods.

In this study, in an effort to best reflect real traffic scenarios in real networks as well as newer means of attack, we chose the CICIDS2017 dataset (Canadian Institute for Cybersecurity) [30] which contains benign traffic and up-to-date common attack traffic representative of a real network environment. This dataset constructs the abstract behavior of 25 users based on HTTP, HTTPS, FTP, SSH, and e-mail protocols to accurately simulate a real network environment. The data capture period was from 9 a.m. on July 3, 2017, to 5 p.m. on July 7, 2017; a total of 51.1 g data flow was generated over this five-day period. The attack traffic collected includes eight types of attack: FTP-Patator, SSH-Patator, DoS, Heartbleed, Web Attack, Infiltration, Botnet, and DDoS. As shown in Table 1, the attacks were carried out on Tuesday, Wednesday, Thursday, and Friday morning and afternoon. Normal traffic was generated throughout the day on Monday and during the nonaggressive period from Tuesday to Friday. The data type for this dataset is a pcap file.

After acquiring the dataset, we analyzed the original data and selected seven types of data for subsequent assessment according to the amount of data and its noise rate. They are Normal, FTP-Patator, SSH-Patator, DoS, Heartbleed, Infiltration, and PortScan.

3.2. Network Traffic Segmentation Method. The format of the CICIDS2017 dataset is one pcap file per day. These pcap files contain a great deal of information, which is not conducive to training the machine. Therefore, the primary task of traffic classification based on machine learning is to divide continuous pcap files into several discrete units according to a certain granularity. There are six ways to slice network traffic: by TCP, by connection, by network flow, by session, by service class, and by host. When the original traffic data is segmented according to different methods, it splits into quite different forms, so the selected network traffic segmentation method markedly influences the subsequent analysis.

We adopted a session sharding method in this study. A session is any packet that consists of a bidirectional flow, that is, any packet that has the same quad (source IP, source port, destination IP, destination port, and transport layer protocol) and interchangeable source and destination addresses and ports.

3.3. Data Preprocessing. Data preprocessing begins with the original flow, namely, the data in pcap format, for formatting the model input data. The CICIDS2017 dataset provides an original dataset in pcap format and a CSV file detailing some of the traffic. To transform the original data into the model input format, we conducted time division, traffic

Table 1: CICIDS2017 dataset.

Date	Type	Size
Monday	Normal	11.0GB
Tuesday	Normal + Force + SFTP + SSH	11.0GB
Wednesday	Normal + Dos + Heartbleed Attacks	13GB
Thursday	Normal + XSS + Web Attack + Infiltration	7.8GB
Friday	Normal + Botnet + PortScan + DDoS	8.3GB

segmentation, PKL file generation, PKL file labeling, matrix generation, and one_hot encoding. A flow chart of this process is given in Figure 1.

Step 1 (time division). Time division refers to intercepting the pcap file of the corresponding period from the original pcap file according to the attack time and type [30]. The input format is, again, a pcap file; the output format is still a pcap file. The time periods corresponding to the specific type and the size of the file are shown in Table 2.

Step 2 (traffic segmentation). Traffic segmentation refers to dividing the pcap file obtained in Step 1 into corresponding sessions by sharding according to the IP of attack host and victim host corresponding to each time period [31]. The specific process is shown in Figure 2. This step involves shredding the pcap file of Step 1 into the corresponding flow using pkt2flow [31], which can split the pcap package into the flow format (i.e., the original pcap package is divided into different pcap packages according to the flow with different five-tuples). Next, the pcap package is merged under the premise that the source and destination are interchangeable. Finally, the pcap package of Step 1 is divided into sessions.

Step 3 (generate the PKL file) and Step 4 (tag the PKL file). As shown in Table 1, the pcap file is still large in size after extraction; this creates a serious challenge for data reading in the model. To accelerate the data reading process, we packaged the traffic using the pickle tool in Python. We use PortScan type traffic as an example of the packaging process here. In this class, many sessions are generated after Step 2, each of which is saved in a pcap file. We generated a label of the corresponding attack type for each session. Each session contains several data flows, and each data flow contains an n_1 packet. We then saved the n_2 sessions in a PKL file to speed up the process of reading the data. n_1 can be changed as needed; according to the experimental results, we finally selected the best value, that is, $n_1 = 8$. The value of n_2 can be calculated by formula (1). In this case, we packaged each type of sessions into a PKL file. The structure of the entire PKL file is shown in Figure 3.

$$n_2 = \frac{\text{total number of packets of this type}}{n_1}. \quad (1)$$

Step 5 (matrix generation). The input of the model must have a fixed length, so the next step is to unify the length of each session. The difference between each attack is mainly in the header, so we dealt with the packet according to the uniform length of PACKET_LEN bytes; that is, if the packet length was greater than PACKET_LEN, then bytes were intercepted, and if the packet length was less than PACKET_LEN, bytes were filled with -1. Each session is then divided into a matrix MAX_PACKET_NUM_PER_SESSION∗PACKET_LEN. According to the results of our experiment, we finally chose MAX_PACKET_NUM_PER_SESSION as 8 and PACKET_LEN as 40.

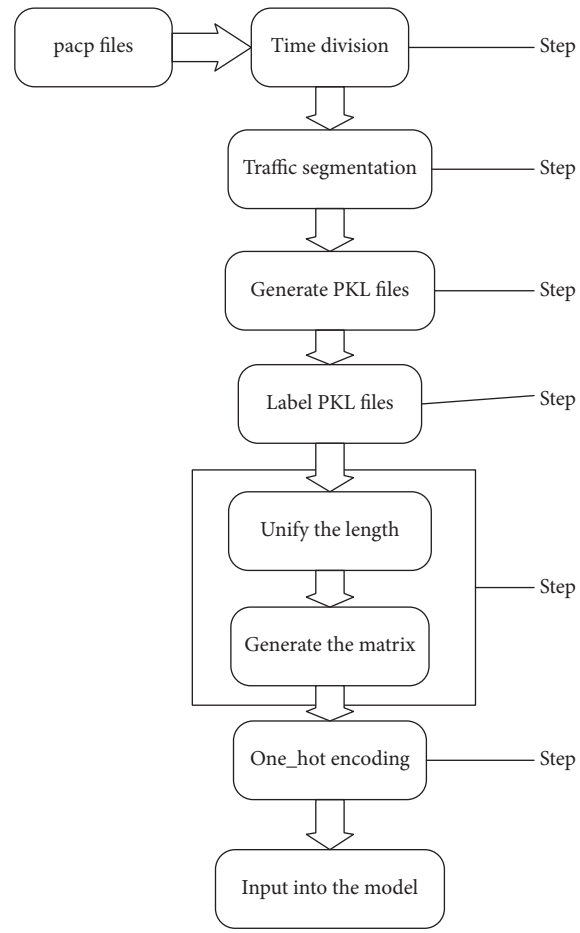

Figure 1: Data preprocessing process.

Table 2: Distribution of network traffic periods in CICIDS2017 dataset.

Attack	Time	Size
Normal	Monday	11 GB
FTP-Patator	Tuesday (9 : 20–10 : 20)	12 MB
SSH-Patator	Tuesday (14 : 00–15 : 00)	25 MB
DoS	Wednesday (9 : 47–11 : 23)	2.3 GB
Heartbleed	Wednesday (15 : 12–15 : 32)	79 MB
Infiltration	Thursday (14 : 19–15 : 45)	21 MB
PortScan	Friday (13 : 55–14 : 35)	31 MB

Step 6 (one_hot encoding). To effectively learn and classify the model, the data from Step 5 are processed by one_hot encoding to convert qualitative features into quantitative features:

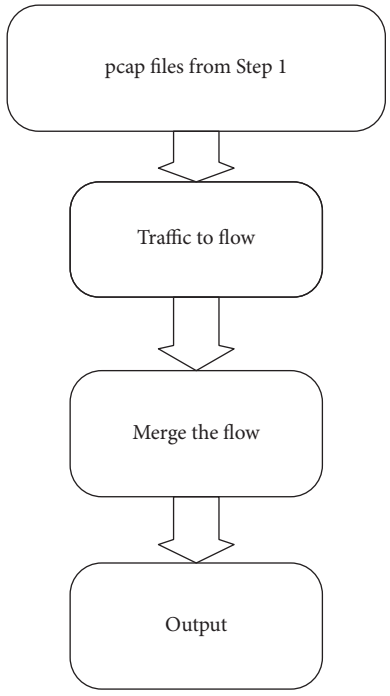

Figure 2: Traffic segmentation process.

$$ohe_i(B, num) = \begin{cases} num, & B = i, \\ 0, & B \neq i, \end{cases}$$
$$OHE_j\left(ohe_{j1}, \ldots, ohe_{j(\text{Length})}\right) = ohe_1 \oplus \ldots \oplus ohe_{\text{Length}},$$
$$\text{Input}_k = \begin{pmatrix} OHE_1 \\ OHE_2 \\ \ldots \\ OHE_N \end{pmatrix}, \tag{2}$$

where B is a byte in the data packet; num is the number used for encoding in one_hot encoding. In the model implementation, $num = 1$, ohe_i is a bit in the one_hot encoding of a byte, $\oplus$ is the series notation, and OHE_j is the one_hot encoding of a byte.

4. DL-IDS Architecture

This section introduces the traffic classifier we built into DL-IDS, which uses a combination of CNN and LSTM to learn and classify traffic packets in both time and space. According to the different characteristics of different types of traffic, we also used the weight of classes to improve the stability of the model.

The overall architecture of the classifier is shown in Figure 4. The classifier is composed of CNN and LSTM. The CNN section is composed of an input and embedded layer, convolution layer 1, pooling layer 2, convolution layer 3, pooling layer 4, and full connection layer 5. Upon receiving a preprocessed PKL file, the CNN section processes it and returns a high-dimensional package vector to the LSTM section. The LSTM section is composed of the LSTM layer 6, LSTM layer 7, full connection layer 8, and the OUTPUT layer. It can process a series of high-dimensional package vectors and output a vector that represents the probability that the session belongs to each class. The Softmax layer outputs the final result of the classification according to the vector of probability.

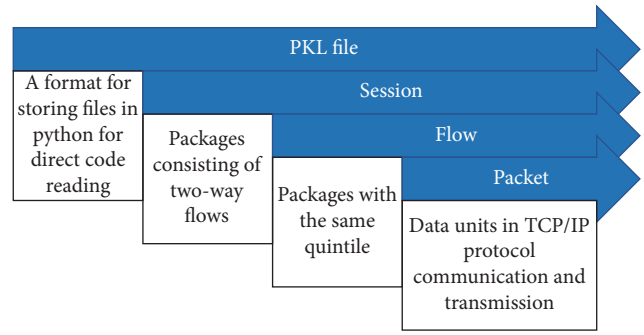

Figure 3: Structure of entire PKL file.

4.1. CNN in DL-IDS. We converted the data packets obtained from the preprocessed data into a traffic image. The so-called traffic image is a combination of all or part of the bit values of a network traffic packet into a two-dimensional matrix. The data in the network traffic packet is composed of bytes. The value range of the bytes is 0–255, which is the same as the value range of the bytes in images. We took the x byte in the header of a packet and the y byte in the payload and composed them into a traffic image for subsequent processing, as discussed below.

As mentioned above, the CNN section is composed of input and embedded layers, convolution layer 1, pooling layer 2, convolution layer 3, pooling layer 4, and full connection layer 5. We combined convolution layer 1 and pooling layer 2 into Combination I and convolution layer 3 and pooling layer 4 into Combination II. Each Combination allows for the analysis of input layer characteristics from different prospective. In Combination I, a convolution layer with a small convolution kernel is used to extract local features of traffic image details (e.g., IP and Port). Clear-cut features and stable results can be obtained in the pooling layer. In Combination II, a large convolution kernel is used to analyze the relationship between two bits that are far apart, such as information in the traffic payload.

After preprocessing and one_hot coding, the network traffic constitutes the input vector of the input layer. In the input layer, length information is intercepted from the ith packet $Pkg_i = (B_1, B_2, \ldots, B_{\text{Length}})$ followed by synthesis S of n Pkgs information set $S = (Pkg_1, Pkg_2, \ldots, Pkg_n)$.

Formula 3 shows a convolution layer, where f is the side length of the convolution kernel. In the two convolution layers, $f = 7$ and $f = 5$. s is stride, p is padding, b is bias, w is weight, c is channel, l is layer, L_l is the size of Z_l, and $Z(i, j)$ is the pixel of the corresponding feature map. Additionally $(i, j) \in \{0, 1, 2, \ldots, L_{l+1}\} L_{l+1} = ((L_l + 2p - f)/s) + 1$.

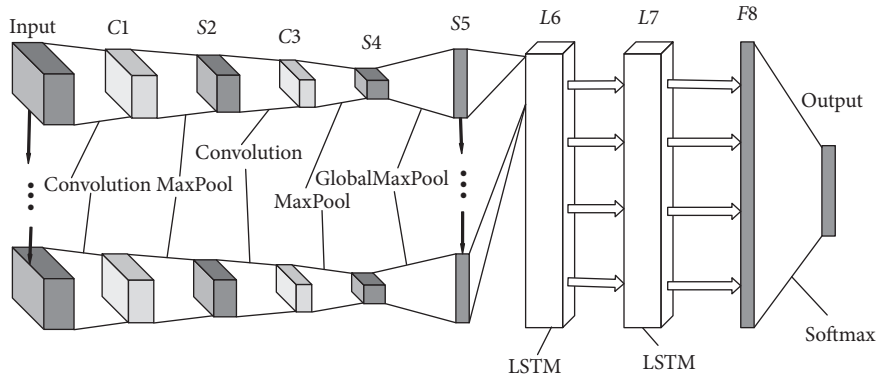

Figure 4: Architecture of DL-IDS.

$$Z^{l+1}(i,j) = \sum_{k=1}^{c} \sum_{x=1}^{f} \sum_{y=1}^{f} \left[Z_k^1 (s*i+x, s*j+y) * w_k^{l+1}(x,y) \right] + b. \tag{3}$$

The convolution layer contains an activation function (formula 4) that assists in the expression of complex features. K is the number of channels in the characteristic graph and A represents the output vector of the Z vector through the activation function. We used sigmoid and ReLU, respectively, after two convolution layers.

$$A_{i,j,k}^1 = f\left(Z_{i,j,k}^1\right). \tag{4}$$

After feature extraction in the convolution layer, the output image is transferred to the pooling layer for feature selection and information filtering. The pooling layer contains a preset pooling function that replaces the result of a single point in the feature map with the feature graph statistics of its adjacent region. The pooling layer is calculated by formula 5, where p is the prespecified parameter. We applied the maximum pooling in this study, that is, $p \longrightarrow \infty$.

$$A_k^l(i,j) = \left[\sum_{x=1}^{f} \sum_{y=1}^{f} A_k^l (s*i+x, s*j+y)^p \right]^{1/p}. \tag{5}$$

We also used a back-propagation algorithm to adjust the model parameters. In the weight adjustment algorithm (formula 6), δ is delta error of loss function to the layer, and α is the learning rate.

$$w^l = w^l - \alpha \sum \delta^l * A^{l-1}. \tag{6}$$

We used the categorical cross-entropy algorithm in the loss function. In order to reduce training time and enhance the gradient descent accuracy, we used the RmsProp optimization function to adjust the learning rate.

After two convolution and pooling operations, we extracted the entire traffic image into a smaller feature block, which represents the feature information of the whole traffic packet. The block can then be fed into the RNN system as an input to the RNN layer.

4.2. LSTM in DL-IDS. Normal network communications and network attacks both are carried out according to a certain network protocol. This means that attack packets must be ensconced in traffic alongside packets containing fixed parts of the network protocol, such as normal connection establishments, key exchanges, connections, and disconnections. In the normal portion of the attack traffic, no data can be used to determine whether the packet is intended to cause an attack. Using a CNN alone to train the characteristics of a single packet as the basis for the system to judge the nature of the traffic makes the data difficult to mark, leaves too much "dirty" data in the traffic, and produces altogether poor training results. In this study, we remedied this by introducing the LSTM, which takes the data of a single connection (from initiation to disconnection) as a group and judges the characteristics of all data packets in said group and the relations among them as a basis to judge the nature of the traffic. The natural language processing model performs well in traffic information processing [32] under a similar methodology as the grouping judgment method proposed here.

The LSTM section is composed of LSTM layer 6, LSTM layer 7, full connection layer 8, and Softmax and output layers. The main functions are realized by two LSTM layers. The LSTM is a special RNN designed to resolve gradient disappearance and gradient explosion problems in the process of long sequence training. General RNN networks only have one tanh layer, while LSTM networks perform better processing timing prediction through their unique forgetting and selective memory gates. Here, we call the LSTM node a cell (C_t), the input and output of which are x_t and h_t, respectively.

The first step in the LSTM layer is to determine what information the model will discard from the cell state. This decision is made through the forgetting gate (formula 7). The gate reads h_{t-1} and x_t and outputs a value between 0 and 1 to each number in the C_{t-1} cell state; 1 means "completely retained" and 0 means "completely discarded." W and b are weight and bias in the neural network, respectively.

$$f_t = \sigma\left(W_f \cdot \left[h_{t-1}, x_t\right] + b_f\right). \tag{7}$$

The next step is to decide how much new information to add to the cell state. First, a sigmoid layer determines which information needs to be updated (formula 8). A tanh layer generates a vector as an alternative for updating (formula 9). The two parts are then combined to make an update to the state of the cell (formula 10).

$$i_t = \sigma\left(W_i \cdot \left[h_{t-1}, x_t\right] + b_i\right), \tag{8}$$

$$\widetilde{C}_t = \tan h\left(W_c \cdot \left[h_{t-1}, x_t\right] + b_c\right), \tag{9}$$

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t. \tag{10}$$

The output gate determines the output of the cell. First, a sigmoid layer determines which parts of the cell state are exported (formula 11). Next, the cell state is processed through a tanh function to obtain a value between −1 and 1 and then multiplied by the output of the sigmoid gate. The output is determined accordingly (formula 12).

$$o_t = \sigma\left(W_o \cdot \left[h_{t-1}, x_t\right] + b_o\right), \tag{11}$$

$$h_t = o_t * \tan h\left(C_t\right). \tag{12}$$

In the proposed model, the feature maps of n data packets in a group of traffic images in a connection serve as the input of the LSTM section. The feature relations between these n data packets were analyzed through the two LSTM layers. The first few packets may be used to establish connections; such packets may exist in the normal data streams,but they may occur in the attack data streams too. The next few packets may contain long payloads as well as attack data. The LSTM finds the groups containing attack data and marks all packets of those whole groups as attack groups.

LSTM layer 6 in DL-IDS has a linear activation function designed to minimize the training time. LSTM layer 7 is nonlinearly activated through the ReLU function. The flow comprises a multiclassification system, so the model is trained to minimize multiclass cross-entropy. We did not update the ownership weight at every step but instead only needed to add the initial weight according to the volume of various types of data.

4.3. Weight of Classes. The data obtained after preprocessing is shown in Table 3, where, clearly, the quantities ("numbers") of different data types are uneven. The number of type 0 is the highest, while those of types 2 and 4 are the lowest. This may affect the final learning outcome of the classification. For example, if the machine were to judge all the traffic as type 0, the accuracy of the model would seem to be relatively high. We introduced the weights of classes to resolve this problem: classes with different sample numbers in the classification were given different weights, class_weight is set according to the number of samples, and class_weight[i] is used instead of 1 to punish the errors in the class [i] samples. A higher class_weight means a greater emphasis on the class. Compared with the case without considering the weight, more samples are classified into high-weight classes.

Table 3: Quantity of data per category in CICIDS2017.

Label	Attack	Num
0	Normal	477584
1	FTP-Patator	11870
2	SSH-Patator	7428
3	DoS	63240
4	Heartbleed	4755
5	Infiltration	64558
6	PortScan	160002

The class weight is calculated via formula 13, where w_i represents the class weight of class i and n_i represents the amount of traffic of class i.

$$w_i = \frac{\sum_{i=0}^{K-1} n_i}{n_i}. \tag{13}$$

When training the model, the weighted loss function in formula 14 makes the model focus more on samples from underrepresented classes. K is the number of categories, y is the label (if the sample category is i, then $y_i = 1$; otherwise $y_i = 0$) and p is the output of the neural network, which is the probability that the model predicts that the category is i and is calculated by Softmax in this model. Loss function J is defined as follows:

$$J = -\sum_{i=0}^{K-1} w_i y_i \log\left(p_i\right). \tag{14}$$

5. Experimental Results and Analysis

We evaluated the performance of the proposed model on the CICIDS2017 dataset using a series of selected parameters: (1) the impact of the length of data packets involved in training; (2) the influence of the number of packets in each flow; (3) the impact of the selected batch size; (4) the effect of the number of units in LSTM; and (5) the influence of the weight of classes. We optimized the DL-IDS parameters accordingly and then compared them against a sole CNN and a sole LSTM. The ratio of Train set, Validation set, and Test set is 18 : 1 : 1.

5.1. Metrics. We adopted four commonly used parameters for evaluating intrusion detection systems: accuracy (ACC), true positive rate (TPR), false positive rate (FPR), and F1-score. ACC represents the overall performance of the model, TPR represents the ratio of the real positive sample in the current positive sample to all positive samples, and FPR represents the ratio of the real negative sample wrongly assigned to the positive sample type to the total number of all negative samples. Recall represents the number of True Positives divided by the number of True Positives and the number of False Negatives. Precision represents the number of positive predictions divided by the total number of positive class values predicted. F1-score is a score of a classifier's accuracy and is defined as the weighted harmonic mean of the Precision and Recall measures of the classifier.

$$\begin{aligned}
\mathrm{ACC} &= \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{FP}+\mathrm{FN}+\mathrm{TN}} * 100\%, \\
\mathrm{TPR} &= \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} * 100\%, \\
\mathrm{FPR} &= \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} * 100\%, \\
\mathrm{Precision} &= \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} * 100\%, \\
\mathrm{Recall} &= \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} * 100\%, \\
\mathrm{F1-score} &= \frac{2 * \mathrm{Precision} * \mathrm{Recall}}{\mathrm{Precision}+\mathrm{Recall}} * 100\%.
\end{aligned} \tag{15}$$

For each type of attack, TP is the number of samples correctly classified as this type, TN is the number of samples correctly classified as not this type, FP is the number of samples incorrectly classified as this type, and FN is the number of samples incorrectly classified as not this type. The definitions of TP, TN, FP, and FN are given in Figure 5.

5.2. Experimental Environment. The experimental configuration we used to evaluate the model parameters is described in Table 4.

5.3. LSTM Unit Quantity Effects on Model Performance. The number of units in the LSTM represents the model's output dimension. In our experiments, we found that model performance is first enhanced and then begins to decline as the number of LSTM units continually increases. We ultimately selected 85 as the optimal number of LSTM units.

5.4. Training Packet Length Effects on Model Performance. Figure 6 shows the changes in ACC, TPR, and FPR with increase in the length of packets extracted during training. As per the training results, model performance significantly declines when the package length exceeds 70. It is possible that excessively long training data packets increase the proportion of data packets smaller than the current packet length, leading to an increase in the proportion of units with a median value of −1 and thus reducing the accuracy of the model. However, the data packet must exceed a certain length to ensure that effective, credible, and scientific content is put into training. This also prevents overfitting effects and provides more accurate classification ability for data packets with partial header similarity. We found that a length of 40 is optimal.

Under the condition that the packet length is 40, the efficiency and performance of the DL-IDS intrusion detection system in identifying various kinds of traffic are shown in Table 5.

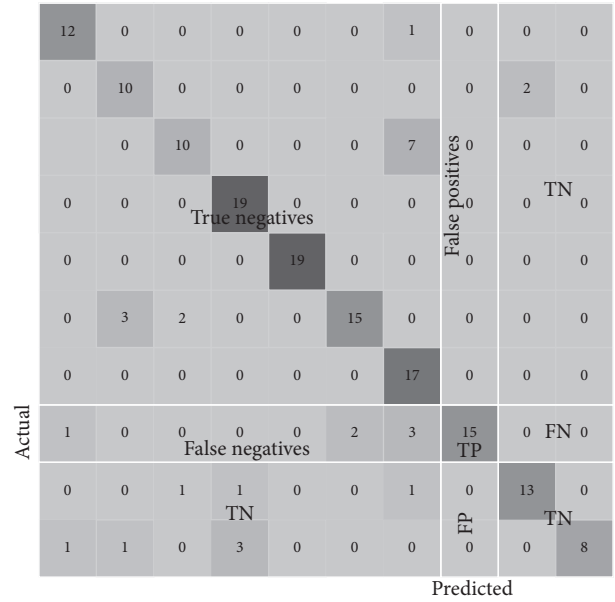

Figure 5: TP, TN, FP, and FN in multiple classifications.

Table 4: Experimental environment.

OS	CentOS Linux release 7.5.1804
CPU	Intel (R) Xeon (R) CPU E5-2620 v3 @ 2.40 GHz
RAM	126 GB
Anaconda	4.5.11
Keras	2.2.2
Python	3.6.5

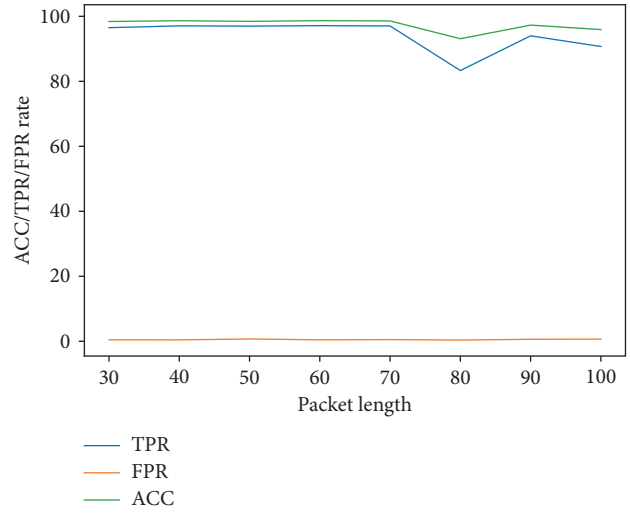

Figure 6: Impact of training packet length on model performance.

5.5. Per-Flow Packet Quantity Effects on Model Performance. As the number of data packets in each flow involved in the training process increases, the features extracted by the model become more obvious and the recognition accuracy

Table 5: Model performance with training packet length of 40.

Label	ACC (%)	TPR (%)	FPR (%)	F1-score (%)
Normal	99.54	99.52	0.00	99.61
FTP-Patator	99.62	92.29	0.27	91.45
SSH-Patator	99.63	87.04	0.00	86.87
Dos	99.61	98.25	0.00	97.87
Heartbleed	99.66	81.12	0.00	81.20
Infiltration	99.55	98.07	0.18	97.54
PortScan	99.54	98.42	0.00	98.71
Overall	98.67	97.21	0.47	93.32

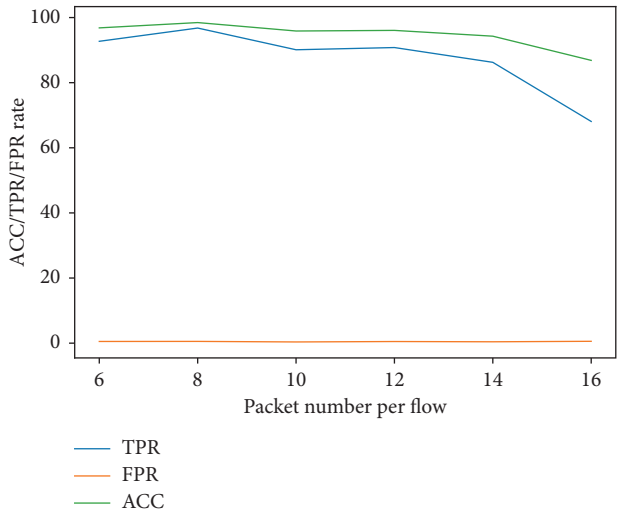

Figure 7: Impact of per-flow packet quantity on model performance.

of the model is enhanced. If this number is too high, however, the proportion of filling data packets increases, thus affecting the model's ability to extract features. Figure 7 shows the impact of the number of packets per flow on model performance. We found that when the number of packets in each flow exceeds 8, the performance of the model declines significantly. We chose 8 as the optimal value of per-flow packet quantity in the network.

5.6. Batch Size Effects on Model Performance. Batch size is an important parameter in the model training process. Within a reasonable range, increasing the batch size can improve the memory utilization and speed up the data processing. If increased to an inappropriate extent, however, it can significantly slow down the process. As shown in Figure 8, we found that a batch size of 20 is optimal.

5.7. Class Weight Effects on Model Performance. Table 6 shows a comparison of two groups of experimental results with and without class weights. Introducing the class weight does appear to reduce the impact of the imbalance of the number of data of various types in the CICIDS2017 dataset on model performance.

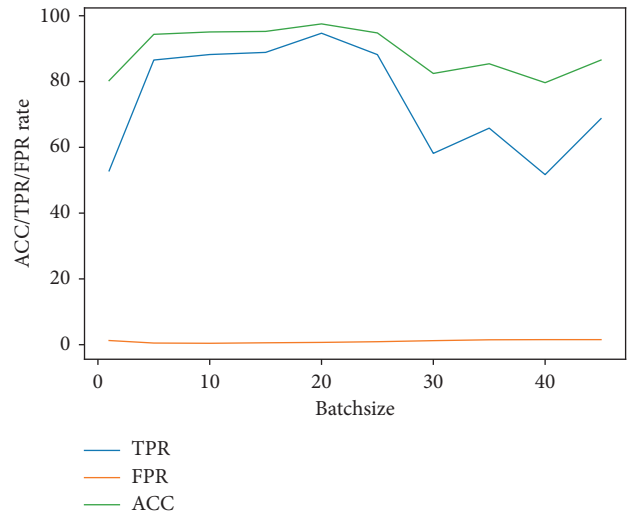

Figure 8: Selected batch size affecting model performance.

Table 6: Effects of applying class weight on model performance.

	Without class weights (%)	With class weights (%)
ACC	97.16	98.58
TPR	93.33	97.04
FPR	0.38	0.52

5.8. Model Evaluation. The LSTM unit can effectively extract the temporal relationship between packets. Table 7 shows a comparison of accuracy between the DL-IDS model and models with the CNN or LSTM alone. The LSTM unit appears to effectively improve the identification efficiency of SSH-Patator, Infiltration, PortScan, and other attack traffic for enhanced model performance, possibly due to the apparent timing of these attacks. Compared to the LSTM model alone, however, adding a CNN further improves the identification efficiency of most attack traffic. As shown in Table 7, the proposed DL-IDS intrusion detection model has very low false alarm rate and can classify network traffic more accurately than the CNN or LSTM alone.

Table 8 shows a comparison of models using CNN and LSTM with traditional machine learning algorithms [33]. The DL-IDS model achieves the best performance among them, with the largest ACC value and the lowest FPR value.

The data input to DL-IDS is raw network traffic. There is no special feature extraction in the model; the training and testing time include the feature extraction time. The traditional machine learning algorithm does not consider data extraction or processing time, so we could not directly compare the time consumption of the various algorithms in Table 8. The training time and testing time of the model were under 600 s and 100 s, respectively, so we believe that the DL-IDS achieves optimal detection effects in the same time frame as the traditional algorithm.

Table 7: CNN, LSTM, and CNN + LSTM results.

Label	CNN (%)				LSTM (%)				CNN + LSTM (%)			
	ACC	TPR	FPR	F1-score	ACC	TPR	FPR	F1-score	ACC	TPR	FPR	F1-score
Normal	99.57	99.56	0.00	99.64	99.56	99.55	0.00	99.63	99.54	99.52	0.00	99.61
FTP-Patator	99.45	91.95	0.27	89.95	99.34	89.27	0.15	91.72	99.62	92.29	0.27	91.45
SSH-Patator	99.53	80.90	0.00	87.22	98.91	79.91	0.00	86.66	99.63	87.04	0.00	86.87
Dos	99.53	98.04	0.00	97.86	98.55	94.85	0.13	89.89	99.61	98.25	0.00	97.87
Heartbleed	99.64	80.08	0.00	80.88	99.63	81.12	0.07	77.69	99.66	81.12	0.00	81.20
Infiltration	99.59	97.91	0.07	97.75	98.84	96.80	1.16	97.17	99.55	98.07	0.18	97.54
PortScan	99.35	97.48	0.00	98.51	99.42	97.99	0.00	94.04	99.54	98.42	0.00	98.71
Overall	98.44	96.46	**0.36**	93.11	96.83	94.21	1.54	90.97	**98.67**	**97.21**	0.47	**93.32**

Table 8: CNN and LSTM models versus traditional machine learning algorithms.

	ACC (%)	TPR (%)	FPR (%)	F1-score (%)
MultinomialNB	72.52	78.20	33.16	52.06
Random Forest	96.08	95.47	3.30	76.71
J48	97.32	96.80	2.17	91.43
Logistic Regression	97.68	94.96	1.47	90.55
DL-IDS	**98.67**	**97.21**	**0.47**	**93.32**

6. Conclusions and Future Research Directions

In this study, we proposed a DL-based intrusion detection system named DL-IDS, which utilized a hybrid of Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) to extract features from the network data flow to analyze the network traffic. In DL-IDS, CNN and LSTM, respectively, extract the spatial features of a single packet and the temporal feature of the data stream and finally fuse them, which improve the performance of intrusion detection system. Moreover, DL-IDS uses category weights for optimization in the training phase. This optimization method reduced the adverse of the number of unbalanced samples of attack types in Train set and improved the robustness of the model.

To evaluate the proposed system, we experimented on the CICIDS2017 dataset, which is often used by researchers for the benchmark. Normal traffic data and some attack data of six typical types of FTP-Patator, SSH-Patator, Dos, Heartbleed, Infiltration, and PortScan were selected to test the ability of DL-IDS to detect attack data. Besides, we also used the same data to test the CNN-only model, the LSTM-only model, and some commonly used machine learning models.

The results show that DL-IDS reached 98.67% and 93.32% in overall accuracy and F1-score, respectively, which performed better than all machine learning models. Also, compared with the CNN-only model and the LSTM-only model, DL-IDS reached over 99.50% in the accuracy of all attack types and achieved the best performance among these three models.

There are yet certain drawbacks to the proposed model, including low detection accuracy on Heartbleed and SSH-Patator attacks due to data lack. Generative Adversarial Networks (GAN) may be considered to overcome the drawback to some degree. Further, combining with some traditional traffic features may enhance the overall model performance. We plan to resolve these problems through further research.

Data Availability

Data will be made available upon request.

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

This research was funded by Beijing Natural Science Foundation under Grant no. 4194086, the National Natural Science Foundation of China under Grant no. 61702043, and the Key R&D Program of Heibei Province under Grant no. 201313701D.

References

[1] M. M. Hassan, A. Gumaei, S. Huda, and A. Ahmad, "Increasing the trustworthiness in the industrial IoT networks through a reliable cyberattack detection model," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, 2020.

[2] F. A. Khan and A. Gumaei, "A comparative study of machine learning classifiers for network intrusion detection," in *Proceedings of the International Conference on Artificial Intelligence and Security*, pp. 75–86, New York, NY, USA, July 2019.

[3] M. Alqahtani, A. Gumaei, M. Mathkour, and M. Maher Ben Ismail, "A genetic-based extreme gradient boosting model for detecting intrusions in wireless sensor networks," *Sensors*, vol. 19, no. 20, p. 4383, 2019.

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

[5] T. Yaqoob, H. Abbas, and M. Atiquzzaman, "Security vulnerabilities, attacks, countermeasures, and regulations of networked medical devices-a review," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3723–3768, 2019.

[6] X. Jing, Z. Yan, X. Jiang, and W. Pedrycz, "Network traffic fusion and analysis against DDoS flooding attacks with a novel reversible sketch," *Information Fusion*, vol. 51, no. 51, pp. 100–113, 2019.

[7] Z. A. Baig, S. Sanguanpong, S. N. Firdous et al., "Averaged dependence estimators for DoS attack detection in IoT networks," *Future Generation Computer Systems*, vol. 102, pp. 198–209, 2020.

[8] Y. Yuan, H. Yuan, D. W. C. Ho, and L. Guo, "Resilient control of wireless networked control system under denial-of-service attacks: a cross-layer design approach," *IEEE Transactions on Cybernetics*, vol. 50, no. 1, pp. 48–60, 2020.

[9] A. Verma and V. Ranga, "Statistical analysis of CIDDS-001 dataset for network intrusion detection systems using distance-based machine learning," *Procedia Computer Science*, vol. 125, pp. 709–716, 2018.

[10] H. Xu, F. Mueller, M. Acharya et al., "Machine learning enhanced real-time intrusion detection using timing information," in *Proceedings of the International Workshop on Trustworthy & Real-Time Edge Computing for Cyber-Physical Systems*, Nashville, TN, USA, 2018.

[11] Y. Wang, W. Meng, W. Li, J. Li, W.-X. Liu, and Y. Xiang, "A fog-based privacy-preserving approach for distributed signature-based intrusion detection," *Journal of Parallel and Distributed Computing*, vol. 122, pp. 26–35, 2018.

[12] H. Xu, C. Fang, Q. Cao et al., "Application of a distance-weighted KNN algorithm improved by moth-flame optimization in network intrusion detection," in *Proceedings of the 2018 IEEE 4th International Symposium on Wireless Systems within the International Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS)*, pp. 166–170, IEEE, Lviv, Ukraine, September 2018.

[13] S. Teng, N. Wu, H. Zhu, L. Teng, and W. Zhang, "SVM-DT-based adaptive and collaborative intrusion detection," *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 1, pp. 108–118, 2018.

[14] J. Liu and L. Xu, "Improvement of SOM classification algorithm and application effect analysis in intrusion detection," *Recent Developments in Intelligent Computing, Communication and Devices*, pp. 559–565, Springer, Berlin, Germany, 2019.

[15] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: a review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[16] T. Ma, F. Wang, J. Cheng, Y. Yu, and X. Chen, "A hybrid spectral clustering and deep neural network ensemble algorithm for intrusion detection in sensor networks," *Sensors*, vol. 16, no. 10, p. 1701, 2016.

[17] Q. Niyaz, W. Sun, A. Y. Javaid, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proceedings of the 9th International Conference on Bio-inspired Information and Communications Technologies*, pp. 21–26, Columbia, NY, USA, December 2015.

[18] A. S. Eesa, Z. Orman, and A. M. A. Brifcani, "A novel feature-selection approach based on the cuttlefish optimization algorithm for intrusion detection systems," *Expert Systems with Applications*, vol. 42, no. 5, pp. 2670–2679, 2015.

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, USA, 2016.

[20] Z. Wang, *The Applications of Deep Learning on Traffic Identification*, pp. 21–26, BlackHat, Las Vegas, NV, USA, 2015.

[21] J. Fan and K. Ling-zhi, *Intrusion Detection Algorithm Based on Convolutional Neural Network*, Beijing Institude of Technology, Beijing, China, 2017.

[22] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, "Long short term memory recurrent neural network classifier for intrusion detection," in *Proceedings of the 2016 International Conference on Platform Technology and Service (PlatCon)*, February 2016.

[23] P. Wu and H. Guo, "LuNET: a deep neural network for network intrusion detection," in *Proceedings of the Symposium Series on Computational Intelligence (SSCI)*, IEEE, Xiamen, ChinaIEEE, Xiamen, China, December 2019.

[24] C. M. Hsu, H. Y. Hsieh, S. W. Prakosa, M. Z. Azhari, and J. S. Leu, "Using long-short-term memory based convolutional neural networks for network intrusion detection," in *Proceedings of the International Wireless Internet Conference*, Springer, Taipei, Taiwan, pp. 86–94, October 2018.

[25] M. Ahsan and K. Nygard, "Convolutional neural networks with LSTM for intrusion detection," in *Proceedings of the 35th International Conference*, vol. 69, pp. 69–79, Seville, Spain, May 2020.

[26] M. M. Hassan, A. Gumaei, A. Ahmed, M. Alrubaian, and G. Fortino, "A hybrid deep learning model for efficient intrusion detection in big data environment," *Information Sciences*, vol. 513, pp. 386–396, 2020.

[27] R. Abdulhammed, H. Musafer, A. Alessa, M. Faezipour, and A. Abuzneid, "Features dimensionality reduction approaches for machine learning based network intrusion detection," *Electronics*, vol. 8, no. 3, p. 322, 2019.

[28] H. Musafer, A. Abuzneid, M. Faezipour, and A. Mahmood, "An enhanced design of Sparse autoencoder for latent features extraction based on trigonometric simplexes for network intrusion detection systems," *Electronics*, vol. 9, no. 2, p. 259, 2020.

[29] V. Ramos and A. Abraham, "ANTIDS: self organized ant based clustering model for intrusion detection system," in *Proceedings of the Fourth IEEE International Workshop on Soft Computing as Transdisciplinary Science and Technology (WSTST'05)*, Springer, Muroran, JapanSpringer, Muroran, Japan, May 2005.

[30] I. Sharafaldin, A. H. Lashkari, and A. Ali, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the The Fourth International Conference on Information Systems Security and Privacy (ICISSP)*, Madeira, Portugal, January 2018.

[31] X. Chen, "A simple utility to classify packets into flows," https://github.com/caesar0301/pkt2flow.

[32] B. J. Radford and B. D. Richardson, "Sequence aggregation rules for anomaly detection in computer network traffic," 2018, https://arxiv.org/abs/1805.03735v2.

[33] A. Ahmim, M. A. Ferrag, L. Maglaras, M. Derdour, and H. Janicke, "A detailed analysis of using supervised machine learning for intrusion detection," 2019, https://www.researchgate.net/publication/331673991_A_Detailed_Analysis_of_Using_Supervised_Machine_Learning_for_Intrusion_Detection.