drones

Article

An Intrusion Detection Model for Drone Communication Network in SDN Environment

Liang Kou [1,†], Shanshuo Ding [1,†], Ting Wu [2,†], Wei Dong [3,*,†] and Yuyu Yin [1,†]

[1] School of Cyberspace, Hangzhou Dianzi University, No. 1, Xiasha No. 2 Street, Qiantang District, Hangzhou 310018, China
[2] Hangzhou Innovation Institute, Beihang University, No. 18, Chuanghui Street, Changhe Street, Binjiang District, Hangzhou 310051, China
[3] National Computer System Engineering Research of China, Future Science and Technology City, Beiqijia Town, Changping District, Beijing 100092, China
* Correspondence: dongwei@ncse.com.cn
† These authors contributed equally to this work.

Abstract: Drone communication is currently a hot topic of research, and the use of drones can easily set up communication networks in areas with complex terrain or areas subject to disasters and has broad application prospects. One of the many challenges currently facing drone communication is the communication security issue. Drone communication networks generally use software defined network (SDN) architectures, and SDN controllers can provide reliable data forwarding control for drone communication networks, but they are also highly susceptible to attacks and pose serious security threats to drone networks. In order to solve the security problem, this paper proposes an intrusion detection model that can reach the convergence state quickly. The model consists of a deep auto-encoder (DAE), a convolutional neural network (CNN), and an attention mechanism. DAE is used to reduce the original data dimensionality and improve the training efficiency, CNN is used to extract the data features, the attention mechanism is used to enhance the important features of the data, and finally the traffic is detected and classified. We conduct tests using the InSDN dataset, which is collected from an SDN environment and is able to verify the effectiveness of the model on SDN traffic. The experiments utilize the Tensorflow framework to build a deep learning model structure, which is run on the Jupyter Notebook platform in the Anaconda environment. Compared with the CNN model, the LSTM model, and the CNN+LSTM hybrid model, the accuracy of this model in binary classification experiments is 99.7%, which is about 0.6% higher than other comparison models. The accuracy of the model in the multiclassification experiment is 95.5%, which is about 3% higher than other comparison models. Additionally, it only needs 20 to 30 iterations to converge, which is only one-third of other models. The experiment proves that the model has fast convergence speed and high precision and is an effective detection method.

Keywords: drones; software defined network; abnormal detection; deep learning; convolutional neural networks; deep auto-encoder; attention mechanism

check for updates

Citation: Kou, L.; Ding, S.; Wu, T.; Dong, W.; Yin, Y. An Intrusion Detection Model for Drone Communication Network in SDN Environment. *Drones* **2022**, *6*, 342. https://doi.org/10.3390/drones6110342

Academic Editor: Vishal Sharma

Received: 8 October 2022
Accepted: 1 November 2022
Published: 4 November 2022

1. Introduction

Drones have long been widely used in the military to accomplish reconnaissance missions or other dangerous tasks through remote control and wireless communication. In recent years, with the maturity and industrialization of drone technology, drones have gradually become an increasingly important part of people's lives. In the field of photography, drones can obtain stunning aerial views at low cost and bring people a new visual experience. In rescue operations, rescuers can use drones to conduct wide-area searches efficiently that will improve rescue efficiency and increase the chances of trapped people being rescued. In the field of transportation, drones can cross over dangerous areas for close-range low-altitude cargo delivery. At the same time, the role of drones in the field of communication is gradually emerging due to its high mobility and dynamic topology [1]. Drones can send data from one point to another with low latency [2]. When the communication infrastructure is damaged by disasters, drones can serve as a bridge between ground users and network nodes [3–5]. SDN decouples the traditional closed network system into a data plane, control plane, and application plane, and makes it possible for the centralized control and management of the network [6]. It separates the control plane of the network from the hardware and replaces the traditional hardware-based control plane with a programmable control plane [7–10]. SDN ensures the reliability of the drone network by closely monitoring the drone network traffic through the controller [3]. The SDN-based drone communication network architecture is shown in Figure 1.

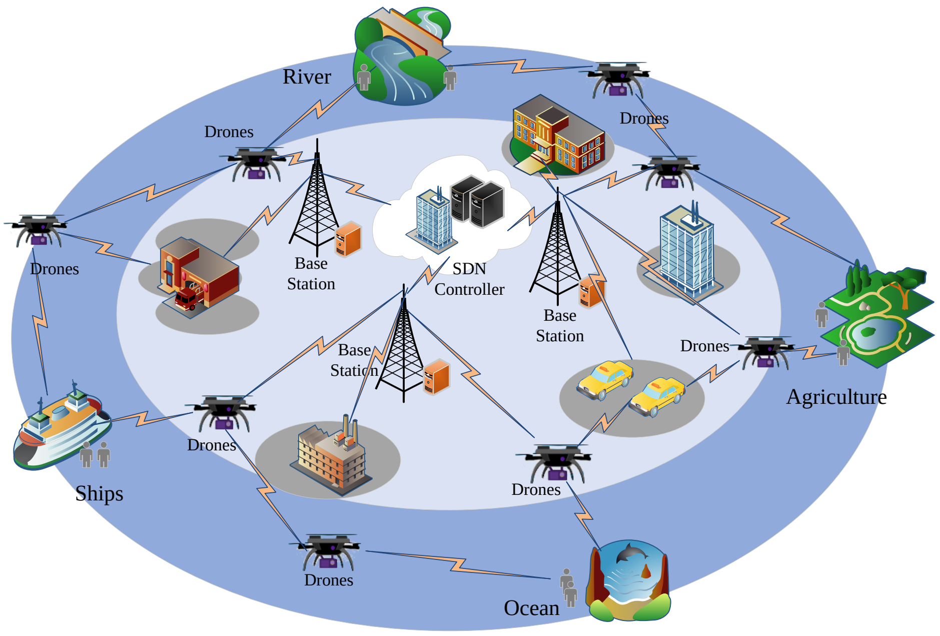


Figure 1. Drone network architecture under SDN.

In remote areas that cannot be covered by cellular base stations, such as farmland, ocean, ships, and other areas, drones can act as drone base stations. People can access the network through drone base stations, and drone base stations can connect with each other for communication. Drone communication networks can flexibly and conveniently provide network services for people. However, due to resource constraints, drone communication networks are extremely vulnerable to physical and cyber attacks [11]. In the past decade, the frequency of network attacks against drone systems has greatly increased [12]. Security and privacy are two of the most critical challenges of drone communication networks [13]. SDNs guarantee the reliability of drone networks, but they cannot guarantee their security because SDNs encounter many challenges. The main challenges are as follows: (1) The controller is the intelligence center of the SDN so it can obtain the global information of the network. At the beginning of the design, the SDN mainly focused on functions such as resource scheduling and rule issuance, but the security issues of SDN did not get enough attention of researchers; (2) Programmability is an important means for SDN to achieve unified network management, but it also provides a shortcut for attackers; (3) In the SDN environment, the openflow switch will forward the unmatched flow request information to the controller that determines the response mode of data flow, which will be very vulnerable to the DoS/DDoS attacks; (4) Attackers can forge the flow rules issued by the SDN controller and bypass various security devices deployed in SDN. Therefore, the security issue of SDN is becoming a key factor restricting the further development and popularization of drone communication networks.

The intrusion detection system (IDS) [14] is a key mechanism to protect and secure the network from malicious attacks. The IDS has two main types: signature-based detection and anomaly-based detection. Signature-based detection methods have a high detection rate for known attacks but cannot deal with new or unknown attacks. The second method can detect all the attacks theoretically. To solve the communication security problem of drone networks in an SDN environment, this paper proposes a novel hybrid abnormal detection model based on an attention mechanism. The main research content of this paper is as follows.

This paper proposes a novel intrusion detection model based on an attention mechanism for the drone network in an SDN environment. The model applies DAE to reduce dimension data and CNN to extract abstract features of data, respectively. The attention mechanism weights and enhances important features to obtain strong representative features. The full connection layer uses softmax to classify and output results. We build a model on the Jupyter Notebook platform in the Anaconda environment. To speed up our training, we use the GTX960 graphics card. We use InSDN dataset to verify the model, and we compare with the other three models: CNN, LSTM, and CNN+LSTM. The results show that the accuracy of our model is 99.7%, which is 0.6% higher than other models, and the accuracy of multiple classification is 95.5%, which is 3% higher than other models. At the same time, it only takes 20 to 30 iterations to basically reach the convergence state, which is one-third of other models.

2. Related Work

Many scholars have studied the security of drone communication networks under SDN. Nawaz et al. [15] studied the self-organizing network, routing protocol, and application scenarios involved in drone communication network. Guerber et al. [16] proposed a secure drone network architecture based on SDN technology, focusing on Ad hoc On-Demand Distance Vector Routing (AODV) routing and the SDN control plane. The advantage of this architecture was accounting for the security and performance of the drones cluster. Altawy et al. [17] investigated physical and cyber threats, key security attributes, and security privacy issues for civilian drones. Sharma et al. [18] proposed an ultra dense cloud-drone network architecture (UDCDN) with great flexibility. The advantages of this architecture were mitigating the severe interference problem of ultra-dense networks and solving the health monitoring and autonomous control problem of UDCDN.

At the same time, there are many scholars who have conducted research on the security issues of SDN technology. Chica et al. [19] conducted a comprehensive survey on security issues under SDN. The survey showed that SDN provides network administrators with more flexible security defense mechanisms because of its network-wide visibility, monitoring capabilities, and flexible network programmability. Other researchers also investigated the security of SDN environment [20,21]. Niyaz et al. [22] proposed a detection system for DDoS attacks in the SDN environment. The system consisted of three modules: traffic collection and flow installer, feature extractor, and traffic classifier. They set up a test environment using a home wireless network to test the system. The experiments showed that SAE as a feature extractor was a beneficial measure for classification. Finally, the model achieved an accuracy of 99.82%. The advantages of this system were complete architecture and high accuracy. Polat et al. [23] used various machine learning methods to detect DDoS attacks in SDN environments and showed that the KNN model achieved the highest accuracy rate of 98.3%. The advantages of this method were lower resource consumption and higher detection efficiency. Malik et al. [24] proposed a hybrid deep learning framework consisting of CNN and LSTM. The model achieved 98.6% detection accuracy for various network attacks, such as xss, botnets, and port scans, which is better than other comparative models. The advantages of this method are that it has strong generalization ability and can detect most attacks. For the security of fog computing devices in an SDN environment, Javanmardi et al. [25] proposed a security-aware task scheduling algorithm called FUPE. The algorithm was based on the fuzzy multi-objective particle swarm optimization method and was mainly used to deal with TCP SYN flooding attacks. The algorithm achieved 98.2% accuracy in the experimental results. At the same time, the algorithm had a 17% maximum improvement in response time and 22% maximum improvement in network utilization over the particle swarm algorithm. The advantages of this algorithm were quick response and high accuracy. Ilango et al. [26] proposed a deep learning scheme FFCNN for LR DoS attacks in the IoT-SDN environment. The method consisted of a feedforward neural network (FFNN) and a convolutional neural network (CNN). The experimental results showed that the scheme had an AUC value of 0.99 and had good classification performance. The hybrid model had strong generalization ability. Quentin Schueller et al. [27] proposed a layered intrusion detection architecture, where the first layer was flow-based intrusion detection, using support vector machine (SVM) as an anomalous traffic detection algorithm to process all passing traffic and filter out suspicious traffic to be delegated to the second layer. The second layer was a signature-based intrusion detection system, where suspicious traffic was detected at the packet level. Due to its two-layer detection structure, it largely ensured a high detection rate and a low false alarm rate. However, it brought a disadvantage of relatively low detection efficiency.

Pynbianglut Hadem et al. [28] proposed an SVM-based SDN intrusion detection system, which was trained and validated on the NSL-KDD dataset and NSL-KDD 9 features subset dataset, respectively. The accuracy rates were 95.98% and 87.74%. However, because it used the KDD dataset and the KDD dataset is a common network traffic dataset, its application to SDN platforms might lead to compatibility issues. Yubo Zhai et al. [29] proposed the use of random forest for intrusion detection with a 98% detection rate. Admilson de Ribamar Lima Ribeiro et al. [30] used the OutlierDenStream algorithm for intrusion detection under SDN with an accuracy of 97.83%. However, it was only for DDoS attacks and did not have a strong generalization capability. Marcos V.O. Assis et al. [31] proposed the use of multilayer GRU for DDoS attack detection and intrusion detection in an SDN environment. Qin et al. [32] proposed the use of LSTM to detect anomalous traffic with high accuracy. Meliboev Azizjon et al. [33] proposed the use of one-dimensional CNN for intrusion detection on unbalanced data. By comparing with other models such as LSTM, it showed that 1D CNN performs well on unbalanced data. MS Elsayed et al. [34] proposed a model that combines CNN and LSTM for intrusion detection. First, spatial features were extracted using a multi-layer CNN structure, then temporal features were extracted using a multi-layer LSTM structure, and finally the classification result were output. On the inSDN dataset, its prediction accuracy was 93.18% for positive samples and 97.60% for negative samples. The disadvantage was that the detection rate of positive samples is not high enough. Pengpeng Ding [35] proposed a model combining CNN and self-attentive mechanism. The data were first extracted by one- and two-dimensional CNNs for feature extraction, respectively, and then the extracted features of different dimensions were fed into the self-attentive mechanism for feature fusion and finally classified by fully connected layers. It was validated on the UNSW-NB15 dataset with an accuracy of 95.64%. Nisha Ahuja [36] proposed a detection model combining stacked auto-encoder and multilayer perceptron for DDoS attack detection with 99.75% detection accuracy, which proved the effectiveness of the model in DDoS attack detection. The disadvantage was that the model only targets DDoS attacks.

The average detection accuracy of existing intrusion detection systems is between 94% and 95%. The detection rate of some intrusion detection systems has reached about 99%. However, these systems have obvious pertinence such as DDoS attacks. The model proposed in this paper can distinguish malicious traffic and normal traffic with 99% probability, whether the malicious traffic comes from DDoS attacks, Probe attacks, or other attacks. At the same time, it can subdivide the types of malicious traffic with 95% accuracy. Table 1 shows a comparison between the work we do and the work done by others.

Table 1. Characteristics and accuracy comparison of intrusion detection models.

Algorithm Involved	Targeting Specific Attacks	SDN Dataset	Detection Accuracy
SVM	F	F	95.98%
Random Forest	F	F	98%
KNN	Y	Y	98.3%
FUPE	Y	Y	98.2%
OutlierDenStream	Y	Y	97.83%
GRU	Y	F	97.1%
LSTM	F	F	98%
CNN	F	F	91.2%
SAE+Multilayer perceptron	Y	F	99.75%
CNN+LSTM	F	Y	97.6%
Our model	F	Y	99.6%

3. Proposed Model

3.1. Deep Auto-Encoder

The auto-encoder (AE) [37] consists of an encoder and a decoder. The essence of the encoder and decoder is the hidden layer in the neural network. The parameters of the AE are optimized based on the backpropagation algorithm and the gradient descent algorithm.

The process from the input layer to the hidden layer is called the encoder. The task of the encoder is to convert the given input data $X = (x_1, x_2, \ldots, x_n)$ to a lower-dimensional representation $H = (h_1, h_2, \ldots, h_r)$. The conversion process is represented by Equation (1).

$$h(x) = f\left(W^{\mathrm{T}}x + b\right) \tag{1}$$

where W^{T} is the weight parameter, b is the offset value, f is the activation function, and h is the hidden vector obtained by the encoder.

The decoder is the inverse process of the encoder. The hidden vector is restored to the input vector through the inverse operation by Equation (2), where $(W^*)^{\mathrm{T}}$ is the weight parameter of each layer of the decoder, c is the offset value of each layer of the decoder, and $h(x)$ is the input from the encoder to the decoder.

$$\hat{x} = f\left((W^*)^{\mathrm{T}}h(x) + c\right) \tag{2}$$

The encoder neural network will get the minimum of loss function through the gradient descent algorithm and then obtain the optimal parameters W, b, and c. The gradient descent algorithm is represented by Equation (3), where x_k is the actual tag value, $\hat{x}_k$ is the output tag value, and n is the number of samples.

$$L(W, b, c) = \frac{1}{n}\sum_{k=1}^{n} \quad (x_k - \hat{x}_k)^2 \tag{3}$$

A single auto-encoder is able to learn limited feature variation through a three-layer network of fictitious input layer → hidden layer → output layer, but for classification tasks involving deep features, the shallow data features obtained through such self-encoder structures often increase the computational effort of subsequent classification tasks, so a deep auto-encoder can be used to extract data features and learn the original data layer by layer multiple representations. The structure of the deep auto-encoder is shown in Figure 2. Each layer is based on the expression of the underlying layer. The extracted features can be more abstract and more suitable for complex classification tasks.

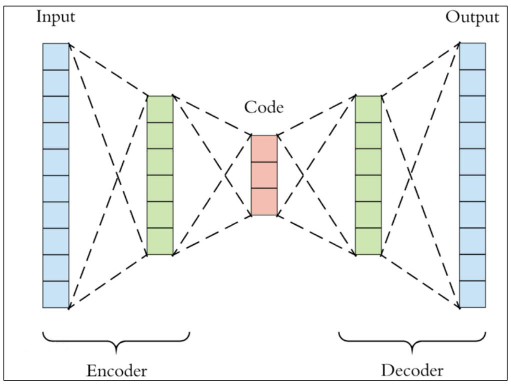


Figure 2. The Encoder and decoder processes of the deep auto-encoder.

The expression x represents the input data, and the initial feature expression h_1 is obtained after being processed by the hidden layer. The idea of the deep auto-encoder is to increase the number of hidden layers of the encoder and decoder so that the subsequent hidden layers continue to extract more abstract features on h_1. After completing the training, the output of the encoder will be a good representative of the original data.

3.2. Convolutional Neural Networks

CNN [38] is a typical deep learning algorithm. It is a feedforward neural network with a deep structure that includes convolution calculations. CNN is mainly composed of three layers: an input layer, a hidden layer, and an output layer. The hidden layer is the key component of CNN. It includes a convolutional layer, a pooling layer, and a fully connected layer.

Convolutional layer: This layer is used to extract features from the input data. It has a few convolution kernels. Each element of the convolution kernel corresponds to a weight and a bias. In a convolutional layer, the feature vector of the previous layer is convolved with the convolution kernel through the activation function to obtain the output vector. The convolution process can enhance the feature of the original data and reduce the effect of noise; x^l is the output of the lth layer, and x_j^l is the jth feature vector of the lth convolutional layer. The value of x_j^l can be calculated by Equation (4).

$$x_j^l = f\left(\sum_{i \in M_j} x_j^{l-1} \otimes k_{ij}^l + b_j^l\right) \tag{4}$$

where M_j is the selection set of input feature vectors, k_{ij}^l is jth convolution kernel parameter of the input feature i, $\otimes$ is the convolution operation, b_j^l is the additive offset, and the Rectified Linear Unit(ReLU) is used as the activation function. ReLU can overcome the vanishing gradient problem.

Pooling layer: The pooling layer usually follows the convolutional layer. It is used for the second feature extraction operation. It can effectively avoid the overfitting problem and strengthen the robustness of the network. The pooling layer performs statistical calculations on the output features from the convolution layer to obtain the statistical probability features instead of the original features. The pooling layer is responsible for downsampling the input vector, which is shown as Equation (5).

$$x_j^l = f\left(\beta_j^l down\left(x_j^{l-1}\right) + b_j^l\right) \tag{5}$$

where the function *down* is responsible for downsampling the jth vector in $l-1$th layer; β_j^l and b_j^l are the multiplicative bias and additive bias, respectively.

Fully connected layer: The fully connected layer in the convolutional neural network is similar to the hidden layer in the traditional feedforward neural network. The fully connected layer is located in the last part of the hidden layer of the convolutional neural network and only transmits signals to other fully connected layers. The feature map is expanded into a vector and loses the spatial topology in the fully connected layer. Each neuron of the fully connected layer is connected to the neuron of the feature vector of the previous layer one by one, and the output of each neuron is expressed by Equation (6).

$$h_{w,b}(x) = f\left(W^T + b\right) \tag{6}$$

where x is the input of the neuron, $h_{w,b}(x)$ is the output of the neuron, w, b are the corresponding weight and offset parameter, W^T is the transpose of the parameter matrix, and b is the offset.

Output layer: The output layer is followed by the fully connected layer. The output layer of the convolutional neural network uses the softmax regression function that is similar to the output layer of the traditional fully connected neural network. For a given test input x, the probability value $p(y = i|j)$ that it belongs to the jth class is calculated. The function is supposed to output one k-dimensional vector, representing k estimated probabilities. The system equation is expressed as Equation (7), where $x^{(i)}$ represents the probability that softmax judges the sample as i, e is introduced to facilitate the subsequent backpropagation derivative calculation, θ represents the layer parameter, k represents the total number of classifications, and j represents other classifications. The output value of the function represents the final probability that the sample belongs to class i.

$$p(y^i = k \middle| x^{(i)}; \theta) = \frac{e^{\theta_k^T x^{(i)}}}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \tag{7}$$

3.3. Attention Mechanism

The attention mechanism [39] is a characteristic mechanism of human vision, in which humans quickly scan an image from the whole picture to obtain the regions that require focused attention and later devote more attention resources in these regions to suppress attention to other unimportant information.

The attention mechanism in deep learning is essentially similar to the human selective visual attention mechanism, and the core goal is also to select the information that is more critical to the current task goal from the many pieces of information. The essence is to obtain a new representation by linear weighting based on the relationships between things.

For example, a text needs to be scored. Each text has a corresponding vector representation and a retrieval library $\{p_i, s_i\}_{i=1}^{K}$, where (p_i, s_i) is a pair of vector representation and rating, now given a text q to be rated, the traditional method to calculate the rating requires calculating the similarity between the query text q and the text inside each retrieval library $sim(p_i, s_i)$; then, we can weight the similarity to get the prediction score α:

$$\alpha = \sum_{i=1}^{K} sim(q, s_i) s_i \tag{8}$$

Correspondingly, attention is given a vector representation q of a query and the corresponding retrieval library $\{k_i, v_i\}_{i=1}^{K}$, where v_i is the vector representation corresponding to keyword k_i. Then, as shown in Figure 3, in order to utilize the knowledge inside q and the retrieval library, the representation of q needs to be transformed as follows, where β is the converted value.

$$\beta = \sum_{i=1}^{K} sim(q, k_i) v_i \tag{9}$$

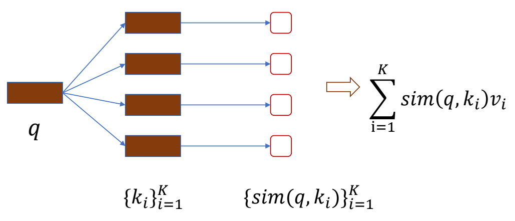


Figure 3. Attention mechanism.

Depending on the method of similarity calculation, there are different kinds of attention, including inner product similarity, cosine similarity, and splicing similarity, where q and k have the same meaning as above; w^T is the transpose of the parameter matrix.

Inner product similarity:

$$sim(q,k) = q^T k \tag{10}$$

Cosine similarity:

$$sim(q,k) = \frac{q^T k}{\|q\|\|k\|} \tag{11}$$

Splicing similarity:

$$sim(q,k) = w^T[q;k] \tag{12}$$

Finally, a set of weights is obtained based on the similarity: $\alpha_i = sim(q,k_i)$, and this weight is normalized.

Attention is to calculate a set of weights using the relationship between things, and then perform a weighted representation to get a new representation, which can be understood as a method of feature transformation.

One of the most commonly used attention mechanisms is feedforward attention, where the query is set to be learnable with the parameter $q = \omega$, and then the key and value inside the retrieval library are set to be the same. The attention mechanism is then obtained using a layer of neural networks to compute attention weights. As shown in the following equation, w is the attention weight, v_i is the keyword vector, and k is the number of keywords.

$$\alpha = Softmax(\omega^T v_1, \omega^T v_2, \ldots, \omega^T v_K) \tag{13}$$

$$Attention(v_1, v_2, \ldots v_k) = \sum_{i=1}^{K} \alpha_i v_i \tag{14}$$

3.4. The DCA Model

Our model is composed of a deep auto-encoder, CNN, and attention, so we name the model the DCA model. The structure of the DCA model proposed in this paper is shown in Figure 4.

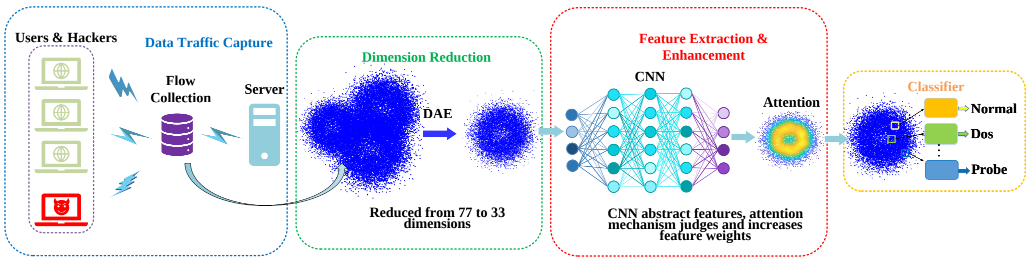

Users & Hackers
Data Traffic Capture
Flow Collection
Server
Dimension Reduction
DAE
Reduced from 77 to 33 dimensions
Feature Extraction & Enhancement
CNN
Attention
CNN abstract features, attention mechanism judges and increases feature weights
Classifier
Normal
Dos
Probe

Figure 4. The DCA model.

In the data preprocessing phase, the first step is to clean the data. Because the dataset is generated from a simulated environment with limited flow-generating devices, Flow ID, Src IP, Src Port, Dst IP, Dst Port, and Timestamp attributes are relatively fixed and not universal. These six features are discarded and 78 features are retained.

For binary classification experiments, the dataset labels need to be modified to 0 and 1, representing normal and abnormal traffic. For the multiclassification experiments, we one-hot coded eight different labels and used 000, 001, 010, 011, 100, 101, 110, and 111 as labels for different attribute flows.

After the numericalization of features, the data are regularized by scaling each sample to unit parity using the L_2 paradigm to avoid the generation of overfitting in the subsequent training process and to reduce the network error. For this dataset, the first 77 dimensions of a sample are data and the 78th dimension is labels. We input the first 77 dimensions of data in the form of $X = (x_1, x_1, x_2, \dots x_{77})$ into the model, labeled as y, and its regularization process is as follows:

$$J'(w; X, y) = J(w; X, y) + \lambda L_2(w) \tag{15}$$

where $L_2(w)$ is:

$$L_2(w) = \|w\|_2^2 = (|w_1|^2 + |w_2|^2 + \dots + |w_n|^2)^{\frac{1}{2}} \tag{16}$$

where w is the vector of weight coefficients, J is the cost function, λ is the parameter controlling the degree of regularization, and $L_2(w)$ is the l_2 parametrization of w. The product of the two is used as the penalty term of the cost function to penalize the high-complexity model.

For the preprocessed data, we assume that one of the samples is $X' = (x'_1, x'_2, x'_3, \dots x'_{77})$. We input it into the deep auto-encoder layer of the model:

$$a_i = \sum_{j=1}^{n} \sigma(W_i^T x_j + b_j) \tag{17}$$

where a_i is the hidden layer output, $i = 1, 2, 3 \dots n$, n is the number of neurons in the hidden layer, W_i^T is the weight matrix of the ith neuron in the hidden layer, and σ is the sigmoid activation function, which is given by:

$$\sigma(a) = \frac{1}{1 + e^{-a}} \tag{18}$$

After processing by the deep auto-encoder, we get the reduced dimensional data matrix: $A = a_{i,j}$, $i \in [0, 343889]$, $j \in [0, 32]$.

We feed the reduced-dimensional data matrix A to the CNN. In the convolutional layer, we assume that the convolution kernel is a matrix $K = k_{p,q}$, and $p * q$ is the convolutional kernel size. We slide the convolution kernel over the matrix A:

$$b_{m,n} = w_{m,n}(\sum a_{i,j} * k_{p,q} + l_{m,n}) \tag{19}$$

Here, $B = b_{m,n}$ is the output matrix after convolution operation by convolution kernel, and $m * n$ is the size of the output matrix after convolution operation. The ReLU activation function will process the matrix B. The equation of the ReLU activation function is:

$$ReLU(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{20}$$

After the maxpooling operation, the matrix is divided into several regions of size 2*1, and the large values are kept to get a new matrix B', which is half the size of the original matrix.

The role of convolution kernels is parameter sharing. A piece of input features of the same size as the convolutional kernel is computed using the same parameters, and different convolutional kernels have different parameters and extract different features. This prevents the problem of parameter explosion in deep convolutional neural networks and greatly reduces the amount of operations.

After the CNN processing, data will be input into the attention mechanism. Suppose the CNN output sequence is $(x_1, x_2, \ldots, x_n)$:

$$C_i = \sum_{j=1}^{n} \alpha_{i,j} x_j \tag{21}$$

where $\alpha_{i,j}$ denotes the attention allocation coefficient of the jth word of the original input at the ith output, and the larger the coefficient, the higher the importance of the current information.

The attention mechanism would use the weighting of elements within each local feature in the feature graph to obtain its weight score. However, this would result in the attention mechanism ignoring the correlation between the local features and the strong information redundancy between the features. In this model, the original features are first downscaled by deep auto-encoder to extract deep abstract features, and then processed by CNN layers to further extract deep features. After the processing of the first two steps, feature redundancy and noise are greatly reduced. The attention mechanism will further weight and strengthen important data features based on the data processed by CNN. This processing flow makes the whole model grasp the key information of features accurately, which is why it has a high recognition accuracy.

Suppose the output vector of the attention mechanism is x. We input x into the classifier and get the classification result. For the binary classification experiment, using the sigmoid function as the classifier, the probability that the sample is a normal flow is:

$$p(y = 1|x;\theta) = h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{22}$$

The probability that the sample is an abnormal flow is:

$$p(y = 0|x;\theta) = 1 - h_\theta(x) = 1 - \frac{1}{1 + e^{-\theta^T x}} \tag{23}$$

The joint sample probabilities are:

$$p(y|x;\theta) = h_\theta(x)^y (1 - h_\theta(x))^{1-y} \tag{24}$$

where $y \in \{0, 1\}$, θ^T is the transpose of the weight parameter matrix, and x is the vector of the input classifier.

After obtaining the sample probabilities, they are fed into the loss function and back-propagated using the chain derivative rule to continuously update the weight parameter θ and the bias parameter b for each layer in order to minimize the cost function. This experiment uses the binary cross-entropy function as the loss function for back propagation:

$$J(\theta) = -\frac{1}{N} \sum_{i=1}^{N} y_i log(p(y_i)) + (1 - y_i) log(1 - p(y_i)) \tag{25}$$

where N is the total number of samples, y is the flow label, and $p(y)$ is the probability that the sample is a normal flow.

For the multiclassification experiment, there are eight classifiers representing eight different types of traffic, and each classifier output is the probability that a sample belongs to that type of traffic. Using the softmax function as a classifier, the process is similar to binary classification, where the eight classifiers output the probability that eight samples belong to that class, and the highest probability is the predicted classification. The probability is derived for each classifier using the following equation:

$$p(y = j|x;\theta) = \frac{e^{\theta_j^T x}}{\sum_{i=1}^{k} e^{\theta_j^T x}} \tag{26}$$

where k is the number of label types and θ is the weight parameter. The cost function used for multiclassification is:

$$J(\theta) = -\frac{1}{N}\Big[\sum_{i=1}^{N}\sum_{j=1}^{k} 1\{y=j\} log \frac{e^{\theta_j^T x}}{\sum_{l=1}^{k} e^{\theta_j^T x}}\Big] \tag{27}$$

where $1\{y=j\}$ is an operation defined with the value of 1 when $y=j$ and 0 when $y \neq j$; N is the total number of samples and k is the number of label types.

4. Exeperimental Evaluation

4.1. InSDN Dataset

Many current intrusion detection systems for SDN use datasets such as KDD99 [40], NSK-KDD [41], etc. These datasets are excellent datasets, but the protocol and network structure used by traditional networks are very different from SDN networks. Therefore, in order to verify the effectiveness of the model in the SDN environment, it is necessary to use the traffic data from the SDN environment.

The dataset used in this paper [42] comes from the SDN virtual environment. The virtual environment is built by multiple virtual machines with an SDN network architecture. The ordinary Ubuntu system are normal users, and the Kali system are attackers to carry out different types of attacks on the SDN network. The dataset contains a total of 343,889 pieces of data; each piece of data contains 84 attributes. There are eight different attributes of traffic in the dataset. Among them, 68,424 are normal traffic and 275,465 are attack traffic. The distribution of data sets is shown in Table 2.

Table 2. Number of different flows.

Flow Type	Quantity
BFA	1405
BOTNET	164
DDoS	121,942
DoS	53,616
Normal	68,424
Probe	98,129
U2R	17
Web-Attack	192

4.2. Simulation Setup

Two sets of comparison experiments are conducted in this section: one set of binary classification experiments and one set of multiclassification experiments. In the two sets of experiments, we compare the traditional single-layer CNN model, the traditional single-layer LSTM model, the CNN+LSTM model, and our proposed DCA model. The experimental environment is an Intel i5 6300HQ CPU, Nvidia GTX960 GPU, 16G memory, Windows 10 system, Python3.6 built under Anaconda3 as a virtual environment, running on a Jupyter Notebook platform.

We choose the Python language as the main model building language. First, the Python language is an interpreted language with simple syntax. It is important for deep learning tasks to quickly build up models and verify model validity. Second, building deep learning models requires constant tuning of details, so programming languages need to be easy to modify. The Python language meets the above needs. At the same time, TensorFlow deep learning libraries are Python programming, so using the Python language is very easy to call.

Because the distribution of normal and abnormal traffic samples in the dataset is unbalanced—normal traffic only accounts for 19.9% of the dataset—it is necessary to carry out stratified sampling division when dividing the training set and test set, so that the distribution of samples in the divided sets is the same as the original dataset to avoid the problems caused by unbalanced data.

In this paper, 80% of the dataset is allocated to the training set and 20% is allocated to the test set. The size of the training set is 275111*78 and the size of the test set is 68778*78. The size of the training dataset is 275111*77, the size of the training label set is 275111*1, the size of the test dataset is 68778*77, and the size of the test label set is 68778*1.

The DCA model is a sequential model whose flow chart and various hyperparameters are given in Figure 5.

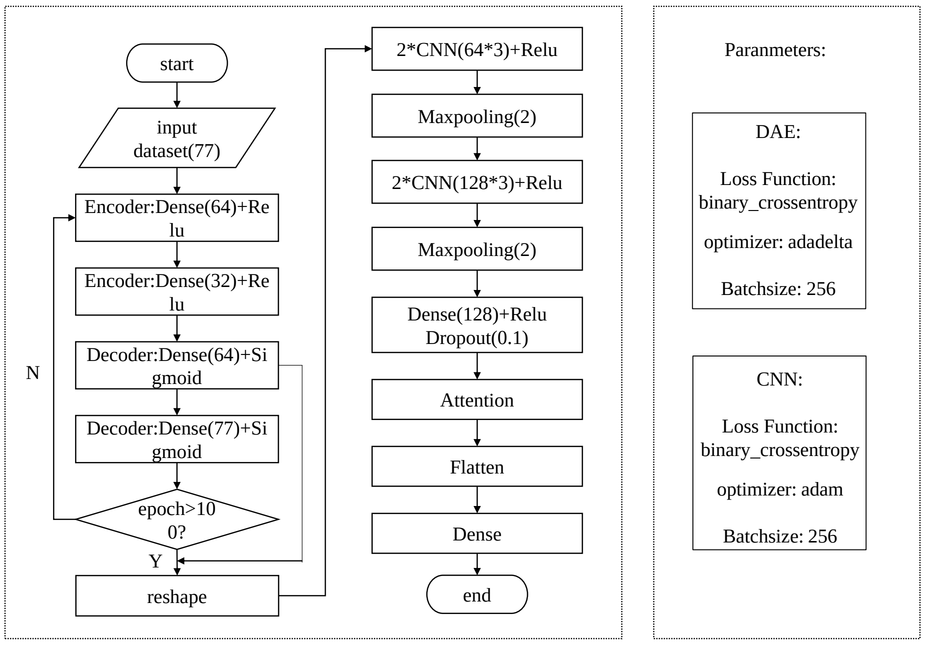


Figure 5. Flow diagram of DCA model.

The deep auto-encoder is unsupervised learning, so the label columns need to be eliminated before training. The hidden layer size of the deep auto-encoder is one of the hyperparameters and needs to be adjusted manually. A good hidden layer output size can preserve the important features of the original data to the greatest extent, and the data restored by the decoder will have lower loss compared with the original data.

In the training process, batch size is one of the important hyperparameters. If batch size is not set, the model will take the whole training set as the unit during training, which will cause a memory explosion problem in the case of a large dataset. By dividing the training dataset into batches of fixed size, the running efficiency of the GPU can be improved by parallelization. Different batch sizes can affect how well the model is trained. We tested the training of batch size at 256, 500, and 1000 for the 100 iterations, and the results are shown in Table 3. Loss represents the optimal loss rate of the training set, and Val Loss represents the loss rate of the validation set.

Table 3. Different batch size and loss.

Batch Size	Loss	Val Loss
256	0.095364	0.095196
500	0.156235	0.165432
1000	0.215263	0.214537

The experimental results show a very large difference in the training results for different batch sizes. Therefore, we choose a batch size of 256 for subsequent experiments.

We set the size of the first hidden layer to 64 and test the effect of the size of the second hidden layer on the dimensionality reduction effect. The results are shown in Table 4.

As you can see from the table, when the hidden layer is set to 32, the loss rate is lower than other sizes, so the hidden layer size is set to 32. We use the trained auto-encoder model to reduce the dimensionality of the dataset.

Table 4. Different hidden layer size and loss.

Hidden Layer Size	Loss	Val Loss
16	0.103622	0.103436
32	0.095364	0.095196
48	0.098497	0.098497

For subsequent experiments, we all use the default parameters of the model for the experiments. Using the encoder of the trained deep auto-encoder, the data input is dimensionalized, and the dimensionalized data is shaped into a three-dimensional form of (343,889, 32, 1) and input into the CNN. The first two convolutional layers with a convolutional kernel size of 64*3 are used for feature extraction, using the ReLU activation function to increase the nonlinearity. The output feature map is pooled by the maxpooling method. The maximum value of each part is selected as the input of the next layer with a size of 2*1 to reduce the size of the feature map and reduce the overhead computation time. After two convolutional layers with 128*3 kernel size and ReLU activation function to extract deeper features, the CNN is connected to a full connection layer with 128 size. The CNN part of the model is now processed. The data are fed into the attention mechanism, where the key information is further processed, and the output is expanded into one-dimensional vector form. In order to prevent overfitting, a dropout operation is added in this step with a parameter of 0.1, which randomly deactivates the neurons with a 10% probability. Finally, the classification is then performed by the classifier.

During the experiments, the loss function converges when the number of iterations is nearly 60 to 70, the loss rate is minimized, and the accuracy has almost stopped growing. Therefore, we set the number of experimental iterations to 100 and compare the optimal results of different models.

4.3. Simulation Metrics

We evaluate the experimental results using the following equations:

$$Accuracy = \frac{TP + TN}{P + N} \tag{28}$$

$$\text{Re}call = \frac{TP}{TP + FN} \tag{29}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{30}$$

$$F - score = 2 * \frac{precision * recall}{precision + recall} \tag{31}$$

where TP (true positive) represents the number of samples that correctly predicted the traffic as normal traffic, TN (true negative) represents the number of samples that correctly predicted the traffic as abnormal traffic, FP (false positive) represents the number of samples that incorrectly predicted the traffic as normal traffic, and FN (false negative) represents the number of samples that incorrectly predicted the traffic as abnormal traffic.

In addition, $Accuracy$ represents the proportion of correctly predicted normal and abnormal traffic in all samples, $Recall$ represents the proportion of all normal traffic that is correctly detected, and $Precision$ represents the proportion of normal traffic among all detected results; $F - score$ synthesizes $Precision$ and $Recall$ as an evaluation index.

4.4. Simulation Scenarios

The dataset we use contains different types of attack traffic, such as DDoS, Probe, etc. In the simulated scenario, the attacker performs various types of attacks on the drone communication network with SDN architecture, while the communication network also has normal communication traffic. For these mixed traffic flows, our proposed model is tested for two tasks. One is to distinguish only normal traffic from malicious traffic, and the other is to analyze the traffic and give traffic labels for each type of traffic.

4.5. Experimental Results of Binary Classification

Both CNN and LSTM belong to the category of deep learning and perform well on text classification tasks. Therefore, our experiments compare the performance of three different models with the same dataset and training times: single-layer CNN, single-layer LSTM, single-layer CNN + single-layer LSTM, and the proposed DCA model, with the number of experimental iterations set to 100. The parameters of the single-layer CNN are identical to those of the CNN part of the model proposed in this paper, and the highest accuracy during training is used for all test models. Table 5 shows the experimental results.

Table 5. Model optimal results.

Model	Accuracy	Recall	Precision	F-Score
CNN	0.991	0.992	0.997	0.995
LSTM	0.994	0.996	0.996	0.996
CNN+LSTM	0.997	0.997	0.999	0.998
DCA	0.997	0.998	0.998	0.998

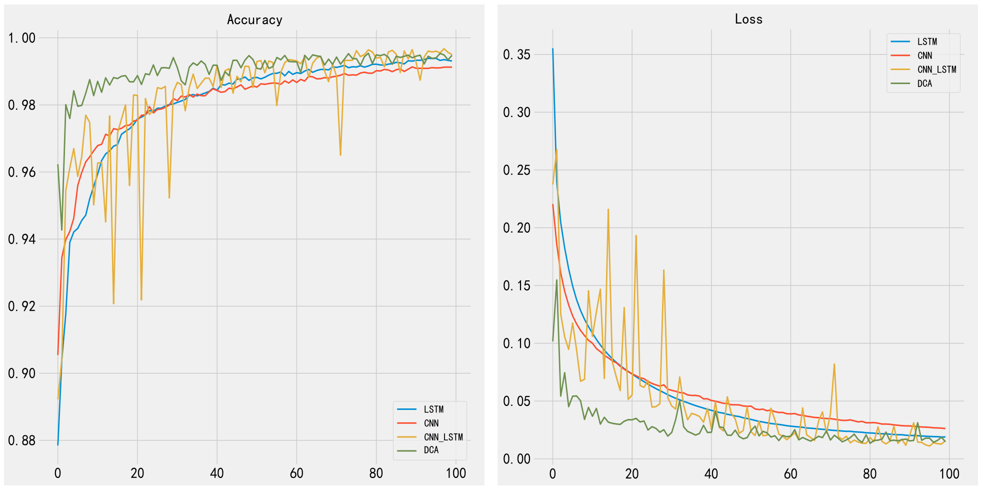
In the results of the four evaluation metrics in Table 6, we choose the model with the best performance in 100 iterations for testing. From the results, we can see that the DCA model is equal to the CNN+LSTM model in terms of accuracy, reaching 0.997, which is better than the other two models. The DCA model outperforms the other three models in terms of recall value, which reaches 0.998. In terms of precision value, the test result of this model is slightly lower than that of the CNN+LSTM model, but better than the other two models. In terms of F-score value, our proposed model is also on par with the CNN+LSTM model, reaching 0.998, which is also better than the other two models.

It is clear that the DCA model is higher than the single-layer CNN model and the single-layer LSTM model in terms of accuracy. However, compared with the CNN+LSTM model, The DCA model seems to have little advantage. This is because the essence of text classification is the transformation of data features into complex functions, which is no different from image classification tasks. The feature complexity of text data is much lower than that in the image processing field. CNN is mainly used in the image processing field and is good at extracting spatial features. The LSTM model is mainly used for processing sequence data and used to extract temporal features. The CNN+LSTM model combines the extraction of both spatial features and temporal features for traffic data, so the accuracy is high when the training is completed. However, it should be noted that it is not comprehensive to judge the merits of the model from the test results of the optimal model alone.

Figure 6a,b show the change curves of the accuracy and loss of each model during 100 iterations. As can be seen from the figures, the accuracy of our proposed model has a huge advantage over the other models in the initial stage of training. Among the other three models, the worst performer is the LSTM model with an initial accuracy below 0.88, the best performer is the single-layer CNN model at approximately 0.91, whereas our proposed model achieves an initial accuracy of approximately 0.97. On the other hand, both single-layer CNN and single-layer LSTM need about 20 iterations to achieve an accuracy of 0.97, and CNN+LSTM needs about 10 iterations.

The DCA model has basically reached convergence at 20 to 30 iterations, whereas the single-layer CNN model and the LSTM model reach convergence at 80 to 90 iterations, and the CNN+LSTM model reaches convergence at 50 to 60 iterations.

Because of its structural design advantages, the DCA model can achieve high accuracy in the initial stage of training and subsequently converge with few iterations. The whole model structure is centered on "extracting key information". From the noise reduction and dimensionality reduction of features by deep auto-encoder to the further extraction of features by CNN and then to the control of key features by the attention mechanism, the whole model goes through three layers of feature processing and constantly discards the redundant noise and weakly correlated features in the data. That is why the DCA model finds the most beneficial feature information quickly for classification and achieves the maximum learning effect with very few iterations.

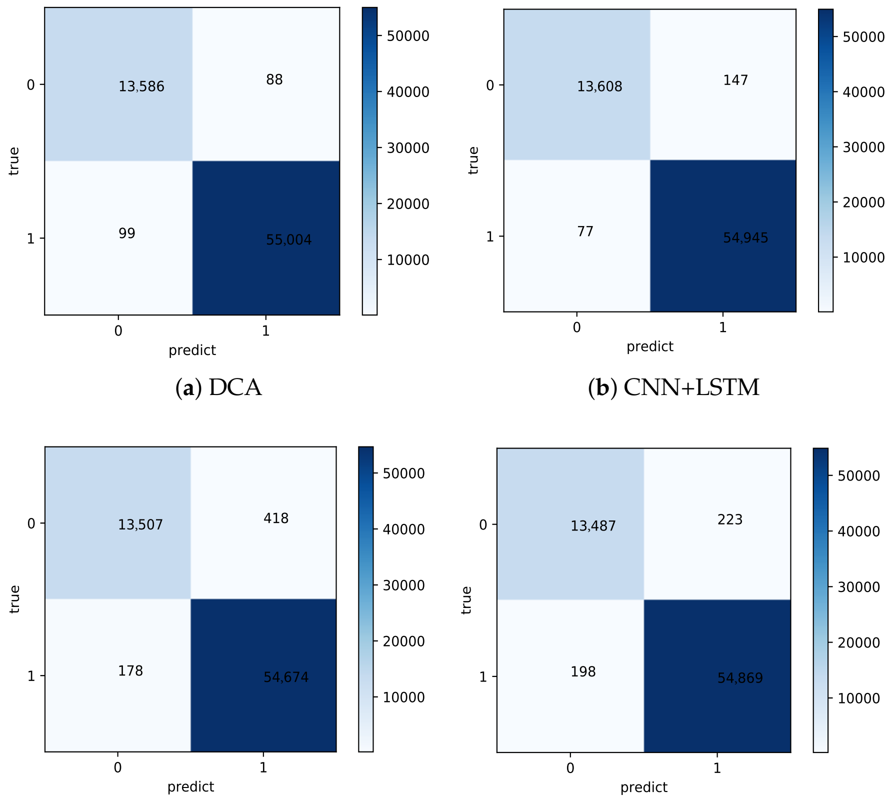


(**a**) Accuracy Curves (**b**) Loss Curves

Figure 6. Binary classification curves.

The single-layer CNN and single-layer LSTM have very smooth training curves due to their simple structures. The CNN+LSTM model may be due to the influence of CNN and LSTM on each other when extracting features, which leads to its unstable extraction of features, and the training curve fluctuates abnormally sharply, although the overall trend is upward, but its stability is worrying. The DCA model curve also fluctuates slightly, but the overall fluctuation is within the controllable range and does not affect its performance.

Figure 7a–d show the confusion matrix generated based on the test results of each model on the test set. The models represented by (a) to (d) are DCA, CNN+LSTM, CNN, and LSTM, respectively. Among the 68,777 test samples, the number of prediction errors for the DCA model was 240, the number of prediction errors for the CNN-LSTM model was 562, and the number of prediction errors for the CNN and LSTM models were 991 and 802, respectively. It can be seen that the DCA model performs significantly better than the other three models on the test set with the lowest number of false positives.



(**a**) DCA (**b**) CNN+LSTM (**c**) CNN (**d**) LSTM

Figure 7. Confusion matrices for different models.

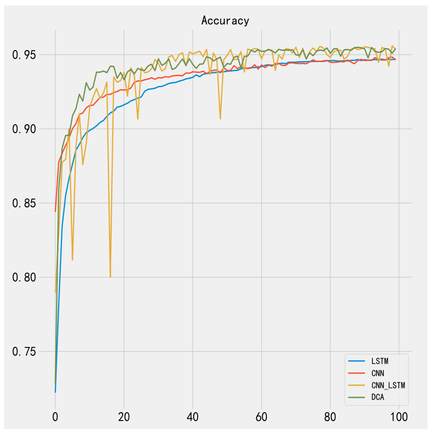
4.6. Experimental Results of Multiple Classification

As shown in Figure 8a,b, the accuracy of all models decreases in the multiclassification experiment compared to the binary classification experiment. The single-layer LSTM and single-layer CNN training curves are very smooth, but the accuracy and loss rates are lower than the other two models. For the CNN+LSTM model, although the final accuracy is not much different from the model proposed in this paper, its model training curve fluctuates drastically, similar to the binary classification experiments. On the other hand, the training time consumes tens of times of the other three models. As shown in Table 6, the CNN+LSTM model takes more than 7 hours to train with the same common parameters, whereas the other three models take only tens of minutes to train.

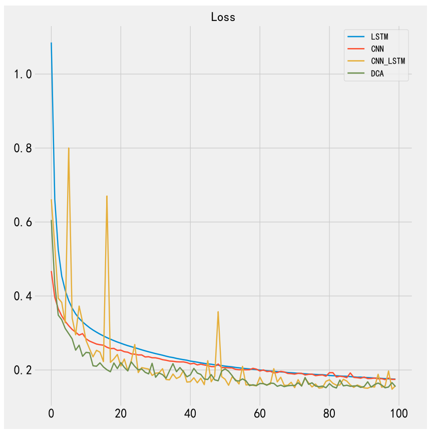
To make the experiment more fair, the hyperparameter batch size of the CNN+LSTM model is increased to reduce its training time. The results are shown in Figure 8c,d.

The accuracy of the CNN+LSTM model decreases significantly, and its training time is 3522 s, which is still much longer than the other three models.

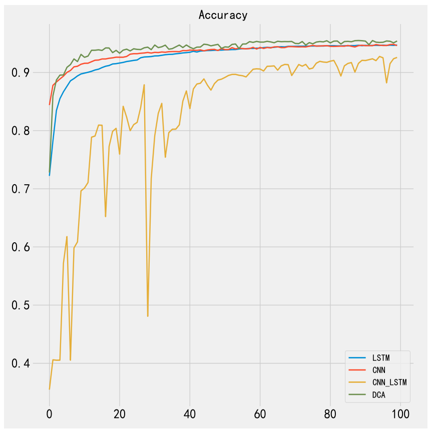
In summary, the DCA model still performs better than the other two traditional models in dealing with multiclassification tasks, whereas the CNN+LSTM model requires a huge time overhead to deal with multiclassification problems with guaranteed accuracy, and the accuracy decreases severely with guaranteed relative efficiency. In summary, the DCA model has a great advantage over the other three models.



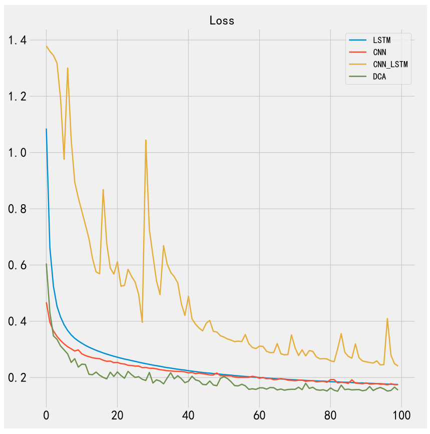
(**a**) Accuracy Curves



(**b**) Loss Curves



(**c**) Accuracy Curves



(**d**) Loss Curves

Figure 8. Multiple classification curves.

Table 6. Multiple classification training time.

Models	**Time Consumed**
CNN	988 s
LSTM	1169 s
DCA	1902 s
CNN+LSTM	27,534 s

5. Conclusions

An intrusion detection algorithm with fast convergence and high accuracy is imperative for drone communication networks due to the limited resources of drones. This paper proposes an intrusion detection model based on the attention mechanism. The model consists of a DAE, CNN, and attention mechanism. DAE reduces invalid features to improve the efficiency of subsequent training and discards invalid features. CNN extracts the abstract features. Attention mechanism gives high weight to important features to increase the impact on the model. We built and trained the model in the Jupyter Notebook platform under the Anaconda environment, and tested the model using InSDN, a dataset entirely from the SDN environment, with CNN, LSTM, and CNN+LSTM, and the DCA model performed well in comparison with these three models. The accuracy of the DCA model was 99.7% in the binary classification experiments, which was 0.6% higher than the other models. The accuracy of the DCA model was 95.5% in the multiple classification experiments, which was 3% higher than the other models. The number of iterations required for the DCA model to reach the convergence state is approximately 20 to 30 rounds of training, which is one-third of that of the other models. The size of the dataset used in this paper is about 350,000 entries. The increase in the number of drones and the development of SDN technology will inevitably cause a dramatic increase in network traffic, and the network data traffic will be much higher than 350,000 entries. This will challenge various existing abnormal traffic detection techniques, and the too slow convergence speed and too long training period as well as the lower accuracy rate will be the reason for elimination. Therefore, based on the excellent performance of the DCA model in the experiments, we have reasons to believe that the model can meet the upcoming challenges. Future work focuses on how to use traffic to update network model parameters in real time and improve the accuracy of the model in multiclassification tasks.

Author Contributions: Conceptualization, L.K., S.D. and T.W.; methodology, L.K. and S.D.; software, S.D.; validation, S.D.; formal analysis, L.K. and S.D.; investigation, L.K., W.D. and Y.Y.; writing—original draft preparation, L.K., S.D.; writing—review and editing, L.K. and S.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: [http://aseados.ucd.ie/datasets/SDN/ or http://iotseclab.ucd.ie/datasets/SDN/ (accessed on 31 October 2022)].

Acknowledgments: This work was supported in part by the Key Technology Research and Development Program of the Zhejiang Province under Grant 2022C01125.

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

References

1. Raja, G.; Anbalagan, S.; Ganapathisubramaniyan, A.; Selvakumar, M.S.; Bashir, A.K.; Mumtaz, S. Efficient and secured swarm pattern multi-UAV communication. *IEEE Trans. Veh. Technol.* **2021**, *70*, 7050–7058. [CrossRef]
2. Ullah, H.; Nair, N.G.; Moore, A.; Nugent, C.; Muschamp, P.; Cuevas, M. 5G communication: An overview of vehicle-to-everything, drones, and healthcare use-cases. *IEEE Access* **2019**, *7*, 37251–37268. [CrossRef]
3. Hassija, V.; Chamola, V.; Agrawal, A.; Goyal, A.; Luong, N.C.; Niyato, D.; Yu, F.R.; Guizani, M. Fast, reliable, and secure drone communication: A comprehensive survey. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 2802–2832. [CrossRef]
4. Alkama, D.; Ouamri, M.A.; Alzaidi, M.S.; Shaw, R.N.; Azni, M.; Ghoneim, S.S.M. Downlink Performance Analysis in MIMO UAV-Cellular Communication with LOS/NLOS Propagation Under 3D Beamforming. *IEEE Access* **2022**, *10*, 6650–6659. [CrossRef]
5. Zhang, J.; Zhang, D.; Sun, J. A Vector-Based Approach for Dimensioning Small Cell Networks in Millimeter-Wave Frequencies. *IEEE Trans. Veh. Technol.* **2022**, *71*, 8980–8993. [CrossRef]
6. Kirkpatrick, K. Software-defined networking. *Commun. ACM* **2013**, *56*, 16–19. [CrossRef]
7. Xia, W.; Wen, Y.; Foh, C.H.; Niyato, D.; Xie, H. A survey on software-defined networking. *IEEE Commun. Surv. Tutor.* **2014**, *17*, 27–51. [CrossRef]
8. Wickboldt, J.A.; De Jesus, W.P.; Isolani, P.H.; Both, C.B.; Rochol, J.; Granville, L.Z. Software-defined networking: management requirements and challenges. *IEEE Commun. Mag.* **2015**, *53*, 278–285 [CrossRef]
9. Shu, Z.; Wan, J.; Li, D.; Lin, J.; Vasilakos, A.V.; Imran, M. Security in software-defined networking: Threats and countermeasures. *Mob. Netw. Appl.* **2016**, *21*, 764–776. [CrossRef]
10. Tang, T.A.; Mhamdi, L.; McLernon, D.; Zaidi, S.A.R.; Ghogho, M. Deep learning approach for network intrusion detection in software defined networking. In Proceedings of the 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM), Fez, Morocco, 26–29 October 2016; pp. 258–263.
11. Siddappaji, B.; Akhilesh, K.B. *Role of Cyber Security in Drone Technology*; Springer: Singapore, 2020; pp. 169–178.
12. Abdelmaboud, A. The Internet of Drones: Requirements, Taxonomy, Recent Advances, and Challenges of Research Trends. *Sensors* **2021**, *21*, 5718. [CrossRef]
13. Yahuza, M.; Idris, M.Y.I.; Wahab, A.W.A.; Nandy, T.; Ahmedy, I.B.; Ramli, R. An edge assisted secure lightweight authentication technique for safe communication on the internet of drones network. *IEEE Access* **2021**, *9*, 31420–31440. [CrossRef]
14. Mukherjee, B.; Heberlein, L.T.; Levitt, K.N. Network intrusion detection. *IEEE Netw.* **1994**, *8*, 26–41. [CrossRef]
15. Nawaz, H.; Ali, H.M.; Laghari, A.A. UAV communication networks issues: A review. *Arch. Comput. Methods Eng.* **2021**, *28*, 1349–1369. [CrossRef]
16. Guerber, C.; Larrieu, N.; Royer, M. Software defined network based architecture to improve security in a swarm of drones. In Proceedings of the 2019 International Conference on Unmanned Aircraft Systems (ICUAS), Atlanta, GA, USA, 11–14 June 2019; pp. 51–60.

17. Altawy, R.; Youssef, A.M. Security, privacy, and safety aspects of civilian drones: A survey. *ACM Trans. Cyber-Phys. Syst.* **2016**, *1*, 1–25. [CrossRef]
18. Sharma, N.; Magarini, M.; Jayakody, D.N.K.; Sharma, V.; Li, J. On-demand ultra-dense cloud drone networks: Opportunities, challenges and benefits. *IEEE Commun. Mag.* **2018**, *56*, 85–91. [CrossRef]
19. Chica, J.C.C.; Imbachi, J.C.; Vega, J.F.B. Security in SDN: A comprehensive survey. *J. Netw. Comput. Appl.* **2020**, *159*, 102595. [CrossRef]
20. Ali, S.T.; Sivaraman, V.; Radford, A.; Jha, S. A survey of securing networks using software defined networking. *IEEE Trans. Reliab.* **2015**, *64*, 1086–1097. [CrossRef]
21. Rawat, D.B.; Reddy, S.R. Software defined networking architecture, security and energy efficiency: A survey. *IEEE Commun. Surv. Tutor.* **2016**, *19*, 325–346. [CrossRef]
22. Niyaz, Q.; Sun, W.; Javaid, A.Y. A deep learning based DDoS detection system in software-defined networking (SDN). *arXiv* **2016**, arXiv:1611.07400.
23. Polat, H.; Polat, O.; Cetin, A. Detecting DDoS attacks in software-defined networks through feature selection methods and machine learning models. *Sustainability* **2020**, *12*, 1035. [CrossRef]
24. Malik, J.; Akhunzada, A.; Bibi, I.; Imran, M.; Musaddiq, A.; Kim, S.W. Hybrid deep learning: An efficient reconnaissance and surveillance detection mechanism in SDN. *IEEE Access* **2020**, *8*, 134695–134706. [CrossRef]
25. Javanmardi, S.; Shojafar, M.; Mohammadi, R.; Nazari, A.; Persico, V.; Pescapè, A. FUPE: A security driven task scheduling approach for SDN-based IoT–Fog networks. *J. Inf. Secur. Appl.* **2021**, *60*, 102853. [CrossRef]
26. Ilango, H.S.; Ma, M.; Su, R. Low Rate DoS Attack Detection in IoT-SDN using Deep Learning. In Proceedings of the 2021 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics), Melbourne, Australia, 6–8 December 2021; pp. 115–120.
27. Schueller, Q.; Basu, K.; Younas, M.; Patel, M.; Ball, F. A hierarchical intrusion detection system using support vector machine for SDN network in cloud data center. In Proceedings of the 2018 28th International Telecommunication Networks and Applications Conference (ITNAC), Sydney, NSW, Australia, 21–23 November 2018; pp. 1–6.
28. Hadem, P.; Saikia, D.K.; Moulik, S. An SDN-based Intrusion Detection System using SVM with Selective Logging for IP Traceback. *Comput. Netw.* **2021**, *191*, 108015. [CrossRef]
29. Zhai, Y.; Zheng, X. Random forest based traffic classification method in SDN. In Proceedings of the 2018 International Conference on Cloud Computing, Big Data and Blockchain (ICCBB), Fuzhou, China, 15–17 November 2018; pp. 1–5.
30. Ribeiro, A.R.L.; Santos, R.Y.C.; Nascimento, A.C.A. Anomaly Detection Technique for Intrusion Detection in SDN Environment using Continuous Data Stream Machine Learning Algorithms. In Proceedings of the 2021 IEEE International Systems Conference (SysCon), Vancouver, BC, Canada, 15 April–15 May 2021; pp. 1–7.
31. Assis, M.V.O.; Carvalho, L.F.; Lloret, J.; Proença, M.L., Jr. A GRU deep learning system against attacks in software defined networks. *J. Netw. Comput. Appl.* **2021**, *177*, 102942. [CrossRef]
32. Qin, G.; Chen, Y.; Lin, Y.X. Anomaly detection using LSTM in IP networks. In Proceedings of the 2018 Sixth International Conference on Advanced Cloud and Big Data (CBD), Lanzhou, China, 12–15 August 2018; pp. 334–337.
33. Azizjon, M.; Jumabek, A.; Kim, W. 1D CNN based network intrusion detection with normalization on imbalanced data. In Proceedings of the 2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Fukuoka, Japan, 19–21 February 2020; pp. 218–224.
34. Elsayed, M.S.; Le-Khac, N.A.; Jahromi, H.Z.; Jurcut, A.D. A Hybrid CNN-LSTM Based Approach for Anomaly Detection Systems in SDNs. In Proceedings of the 16th International Conference on Availability, Reliability and Security (ARES 2021), Vienna, Austria, 17–20 August 2021.
35. Ding, P.; Li, J.; Wang, L.; Wen, M.; Guan, Y. HYBRID-CNN: An efficient scheme for abnormal flow detection in the SDN-Based Smart Grid. *Secur. Commun. Netw.* **2020**, *2020*, 8850550. [CrossRef]
36. Ahuja, N.; Singal, G.; Mukhopadhyay, D. DLSDN: Deep learning for DDOS attack detection in software defined networking. In Proceedings of the 2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence), Noida, India, 28–29 January 2021; pp. 683–688.
37. Ng, A. Sparse autoencoder. *CS294A Lect. Notes* **2011**, *72*, 1–19.
38. Fukushima, K.; Miyake, S. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and Cooperation in Neural Nets*; Springer: Berlin/Heidelberg, Germany, 1982; pp. 267–285.
39. Mnih, V.; Heess, N.; Graves, A. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2014; pp. 2204–2212.
40. University of California at Irvine. UCI KDD Archive. Available online: http://kdd.ics.uci.edu/ (accessed on 9 September 2005).
41. Tavallaee, M.; Bagheri, E.; Lu, W.; Ghorbani, A.A. A detailed analysis of the KDD CUP 99 data set. In Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, Ottawa, ON, Canada, 8–10 July 2009; pp. 1–6.
42. Elsayed, M.S.; Le-Khac, N.A.; Jurcut, A.D. InSDN: A novel SDN intrusion dataset. *IEEE Access* **2020**, *8*, 165263–165284. [CrossRef]