Hindawi
Security and Communication Networks
Volume 2017, Article ID 4184196, 10 pages
https://doi.org/10.1155/2017/4184196

Research Article

Network Intrusion Detection through Stacking Dilated Convolutional Autoencoders

Yang Yu, Jun Long, and Zhiping Cai

College of Computer, National University of Defense Technology, Changsha, Hunan 410073, China

Correspondence should be addressed to Zhiping Cai; zpcai@nudt.edu.cn

Received 28 July 2017; Accepted 22 October 2017; Published 16 November 2017

Academic Editor: Wojciech Mazurczyk

Network intrusion detection is one of the most important parts for cyber security to protect computer systems against malicious attacks. With the emergence of numerous sophisticated and new attacks, however, network intrusion detection techniques are facing several significant challenges. The overall objective of this study is to learn useful feature representations automatically and efficiently from large amounts of unlabeled raw network traffic data by using deep learning approaches. We propose a novel network intrusion model by stacking dilated convolutional autoencoders and evaluate our method on two new intrusion detection datasets. Several experiments were carried out to check the effectiveness of our approach. The comparative experimental results demonstrate that the proposed model can achieve considerably high performance which meets the demand of high accuracy and adaptability of network intrusion detection systems (NIDSs). It is quite potential and promising to apply our model in the large-scale and real-world network environments.

1. Introduction

Network intrusion detection techniques are not trivial for cyber security to defend against malicious and suspicious activities [1, 2]. Anomaly-based network intrusion detection systems (ANIDSs) play a critical role in reacting and protecting against an increasing number of damaging threats and attacks. Furthermore, monitoring and analyzing malware traffic behaviors are especially essential tasks for network anomaly detection.

Unfortunately, network intrusion detection techniques are still facing several enormous challenges and problems to detect anomalies effectively [3]. First, with the continual increase of the number and the variety of sophisticated threats and attacks, network intrusion detection systems (NIDSs) produce high false positives or false alarms. Second, classical machine learning approaches used in network intrusion detection have several challenges, such as the paucity of labeled training data and variability of network traffics [4]. These challenges lead to the difficulties to apply conventional machine learning methods in the large-scale and real-world network environments. Additionally, traditional hand-engineered features are neither readily available nor flexible and adaptive for the emerging complex attacks.

Meanwhile, as computer hardware, such as GPUs, owns increasingly computing capabilities, deep learning techniques achieve incredibly impressive results in several research areas. Convolutional neural networks (CNNs) specially have obtained remarkable performance in the field of computer vision, such as object recognition and image classification. The most powerful part of deep learning techniques is learning feature hierarchies from large amounts of unlabeled data. Therefore, deep learning techniques are quite promising to be applied in the network intrusion detection field.

Recently, various deep learning approaches have been applied to the network intrusion detection area, such as restricted Boltzmann machines (RBMs), deep belief networks (DBNs), stacked autoencoders (SAEs), and supervised learning with convolutional neural networks (CNNs). The existing work about the application of deep learning approaches for network intrusion detection is twofold. For one thing, deep learning techniques are utilized to learn or extract valuable features automatically from raw data, which is called feature extraction. These learned features are then fed into classifiers to further complete classification tasks. For another, specific features are firstly extracted according to domain expert

knowledge. Deep learning algorithms mainly play roles of classifiers which take hand-crafted features as input data.

However, there are several problems or limitations with these studies. To begin with, obtaining large amounts of labeled network data and hand-crafted features is pretty costly, let alone other existing problems of customized features, as mentioned previously. In practice, though, getting lots of unlabeled raw network traffic data with little labeled data is relatively easy. Also, the training process of some deep learning methods, such as DBNs and SAEs, consists of unsupervised pre-training [5] and supervised fine-tuning. In this case, large amounts of unlabeled data and little labeled data are, respectively, used in the two training stages. The obvious disadvantage of these fully connected networks is having large number of training parameters because of full connection of units between adjacent layers. As a result, the number of neural network layers is limited, and training process may be very slow. Instead, CNNs reduce the number of parameters through strategies of sparse connectivity and shared weights, but CNNs for supervised learning need labeled data as input. The original motivation of this research is to propose a suitable and effective deep learning approach to bridge the gap unsupervised feature learning and the advantages of CNNs for ANIDSs.

This research aims to construct a novel network intrusion detection model which combines the strengths of unsupervised feature learning and CNNs to extract or learn critical features automatically from large volumes of raw network packets. In this paper, we propose a network intrusion detection model by stacking dilated convolutional autoencoders which actually combines the concepts of self-taught learning [6] and representation learning [7]. The model is evaluated through different classification tasks with malware traffic data which come from diverse malwares. We also observe and discuss the effects of different hyperparameters on evaluation results and find optimal parameter values for the proposed model. The experimental results demonstrate that our model can get remarkable performance and meet the demand of high accuracy and adaptability of NIDSs.

The remainder of this paper is organized as follows. Section 2 introduces recent related work on the application of deep learning approaches for network intrusion detection. Section 3 describes the proposed model and dataset construction. Section 4 presents experimental results and analysis. Section 5 further analyzes the results and discusses limitations of our method and future work. Section 6 concludes the paper.

2. Related Work

In this section, we review a little recent research that is relevant to our work. Deep learning methods used in unsupervised feature learning tasks for network intrusion detection mainly include restricted Boltzmann machines (RBMs), autoencoders, deep belief networks (DBNs), stacked autoencoders, and various variants of these methods.

In most existing studies, the unsupervised deep learning methods for intrusion detection play roles of unsupervised feature extractors to learn abstract features from hand-crafted features. The abstract features are then taken as input data of a classifier, such as the softmax classifier. For example, Fiore et al. (2013) [8] proposed discriminative RBM (DRBM) to learn abstract features from customized features that did not contain information of packet payloads. These learned features were then fed into softmax classifier for the binary classification, namely, normal and anomalous classification. Javaid et al. (2015) [9] used sparse autoencoder and softmax regression on NSL-KDD dataset [10] which is a revised version of the KDD dataset [11]. Similarly, Erfani et al. (2016) [12] combined DBNs and a linear one-class SVM for anomaly detection on various benchmark datasets. Many other studies follow this kind of pattern, namely, taking hand-engineering features which need specific domain knowledge as the input of unsupervised or supervised deep learning methods.

However, very few attempts have been made to use deep learning techniques to learn useful features or good representations from raw network traffics. Wang (2015) [13] used stacked autoencoders for traffic identification from raw network traffic data and achieved impressive high performance. In addition, Wang et al. (2017) [14] transformed 728-dimensional raw traffic data into images and used CNNs with supervised feature learning for malware or botnet traffic classification. Compared with their work, our method can learn feature representations from massive unlabeled data which contain more diverse attack types. Besides, the features our method learned include temporal information, while they only use spatial features of network traffics. Besides, we do not visualize raw traffic data because there exist huge differences on the application and the structure between network traffic data and images. Thus, it would be unsatisfactory to simply visualize the network traffic data to simulate image classification tasks using CNNs regardless of their semantic meanings.

In this paper, we propose a deep learning approach, called dilated convolutional autoencoders (DCAEs), for the network intrusion detection model, which combines the advantages of stacked autoencoders and CNNs. In essence, the model can automatically learn essential features from large-scale and more various unlabeled raw network traffic data consisting of real-world traffics from botnets, web-based malwares, exploits, APTs (Advanced Persistent Threats), scans, and normal traffics.

3. Methodology

In this section, we first introduce our model for network intrusion detection from an overall perspective. Subsequently, the deep learning method used in the model is described in detail. Finally, we briefly present construction of our datasets.

3.1. Model Description. An overview of the training process of the DCAEs-based model is illustrated in Figure 1. Raw network traffic data in the libpcap file format (.pcap) are transformed into numeric vectors through a data preprocessing module. These numeric vectors are training samples of

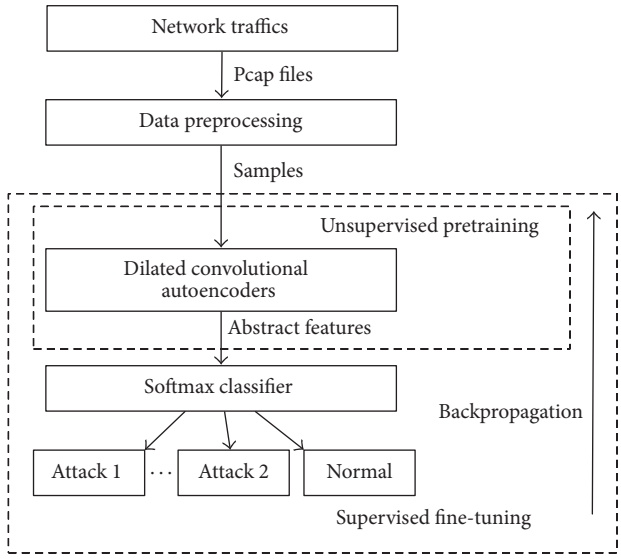

Figure 1: Overview of the training process based on the DCAEs method.

our datasets. The training process is divided into unsupervised pretraining and supervised fine-tuning. In the unsupervised pretraining process, dilated convolutional autoencoders (DCAEs) learn a hierarchy of feature representations from large volumes of unlabeled samples. Afterward, the representations learned from unlabeled data are enhanced by the supervised fine-tuning using the backpropagation algorithm and few labeled samples. Specifically, the neural network is trained as a traditional convolutional neural network without pooling layers using dilated convolutions, as shown in Figure 2. The sample is transformed into the shape of an image for applying dilated convolutions on it. There is only one convolutional layer from a convolutional autoencoder in Figure 2. The early-stopping strategy is used to prevent from over-fitting. In addition, softmax classifier is applied to perform classification task using the abstract features. The use of diverse raw network traffics and unsupervised pretraining makes our model more adaptive and flexible.

3.2. Dilated Convolutional Autoencoders. The architecture of dilated convolutional autoencoders (DCAEs) is pretty similar to classical autoencoders [15]. Figure 3 shows the structure of a dilated convolutional autoencoder. The input is mapped into feature maps through an activation function:

$$\mathbf{h}^k = f\left(\mathbf{x} * \mathbf{W}^k + \mathbf{b}^k\right), \tag{1}$$

where $\mathbf{x}$ is the two-dimensional input reshaped from a numeric vector, $\mathbf{W}^k$ and $\mathbf{b}^k$ are, respectively, a weight matrix and a bias corresponding to the kth feature map h^k. The activation function $f(\cdot)$ in our model is ReLU (Rectified Linear Unit) activation function (i.e., $f(x) = (0, x)$). The symbol $*$ denotes dilated convolution [16] operator. Subsequently, the feature maps of hidden layer are mapped into the reconstruction through a transposed convolution [17]:

$$\widetilde{\mathbf{x}} = f\left(\sum_{k \in H} \mathbf{h}^k * \widetilde{\mathbf{W}}^k + \widetilde{\mathbf{b}}\right), \tag{2}$$

where $\widetilde{\mathbf{x}}$ has the same shape of the input $\mathbf{x}$ and H is a collection of feature maps. The initial values of weight matrix $\mathbf{W}$ and $\widetilde{\mathbf{W}}$ are the same [18]. The learning objective of the dilated convolutional autoencoder is to reduce the difference between the input $\mathbf{x}$ and the reconstruction $\widetilde{\mathbf{x}}$. The cost function in our model is the mean squared error (MSE):

$$L\left(\mathbf{x}, \widetilde{\mathbf{x}}\right) = \frac{1}{n}\sum_{i}^{n}\left(\mathbf{x}_i - \widetilde{\mathbf{x}}_i\right)^2. \tag{3}$$

The DCAEs can be used to construct a deep neural network by stacking multiple DCAEs, which is similar with SAEs [15]. Specifically, the input of the next DCAE is the hidden-layer output of the previous DCAE. The process of stacking DCAEs is greedy layer-wise unsupervised training [19]. One of the advantages of dilated convolutions is that dilated convolution can have a wider range of receptive fields without losing information. This advantage makes it more suitable for text processing.

In sum, the advantages of dilated convolutional autoencoders are as follows. First, the application of dilated convolutions enlarges the layers' receptive fields to learn more global features. Compared with max-pooling, dilated convolutions can protect the input data from information loss. Second, the pretraining process of the DCAEs does not need labeled data, which is more useful in practical applications. Finally, the DCAEs have lesser parameters than fully connected neural networks, such as SAEs. Therefore, the DCAEs are more effective and time-saving than other unsupervised deep learning methods.

3.3. Dataset. In this paper, we performed three kinds of classification tasks on two types of datasets. Table 1 shows the sample distribution for the CTU-UNB dataset and the Contagio-CTU-UNB dataset. The first dataset called the CTU-UNB dataset consists of various botnet traffics from CTU-13 dataset [20] and normal traffics from the UNB ISCX IDS 2012 dataset [21, 22]. The second dataset called the Contagio-CTU-UNB dataset consists of six types of network traffic data. The normal and botnet traffics come from parts of the CTU-UNB dataset. The web-based malware traffics are from the threatglass website [23]. The traffics of exploits, APTs, and Scans come from parts of contagio or deepend research [24]. The general data preprocessing steps are shown in Figure 4. The specific elaboration about data preprocessing and CTU-UNB dataset is presented in our previous work [25].

4. Experimental Results and Performance Analysis

In this section, we first briefly introduce classification metrics for performance analysis. Experimental setup and environments are then described. Finally, we present and analyze some important experimental results.

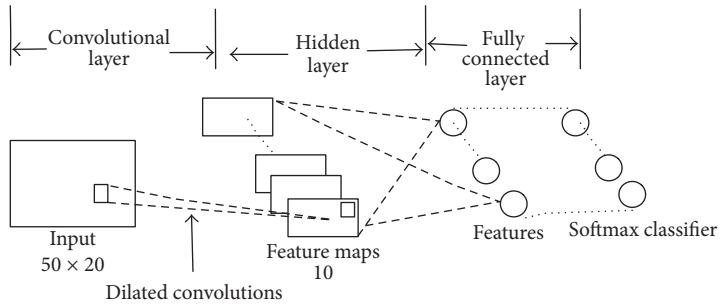

Figure 2: Neural network structure of the fine-tuning process.

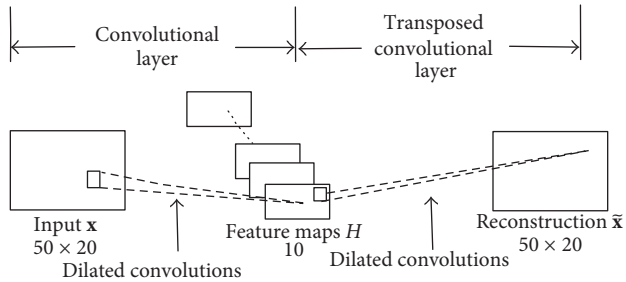

Figure 3: Structure of a dilated convolutional autoencoder.

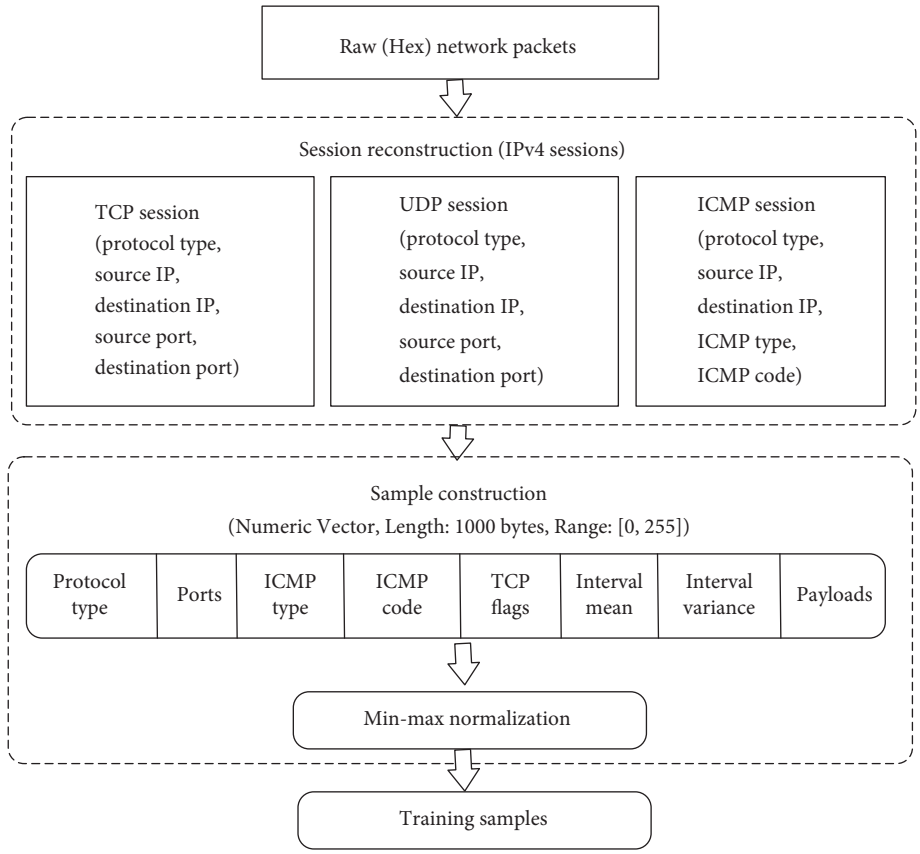

Figure 4: Data preprocessing steps.

Table 1: Sample distribution of the CTU-UNB dataset and the Contagio-CTU-UNB dataset.

CTU-UNB dataset					Contagio-CTU-UNB dataset				
Traffic type		Training set	Validation set	Test set	Total	Traffic type	Training set	Validation set	Test set	Total
Normal		41480	8123	8174	57867	Normal	10812	2053	2061	14926
Bot	Neris	8039	1567	1565	11171	Botnet	10681	2107	2047	14835
	Rbot	6073	1228	1221	8522	Web-based malware	10327	1928	1929	14184
	Virut	18914	3680	3767	26361					
	Menti	217	40	43	300	Exploit	7325	1396	1418	10139
	Sogou	34	5	5	44	APT	1439	242	276	1957
	Murlo	2013	364	358	2735					
	NSIS.ay	4395	903	867	6165	Scan	6466	1274	1269	9009
	Total	39685	7877	7826	55298					
Total		81165	16000	16000	113165	Total	47050	9000	9000	65050

4.1. Classification Metrics and Experimental Setup. Six evaluation metrics were utilized for performance analysis of our experiments. The six metrics are accuracy (AC), precision (P), recall (R), f-measure (F), the receiver operating characteristic (ROC) curve, and the confusion matrix, respectively. Specifically, these metrics are related to four classification functions, namely, true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). In other words, TP and TN separately measure the number of attacks and normal classification correctly. FP and FN represent that the proportion of attacks and normal data that is incorrectly identified, respectively. These four functions can be calculated from the confusion matrix $\mathbf{C}_{i,j}$ whose elements of leading diagonal are the number correctly predicted samples. For example, the element c_{ij} $(i \neq j)$ describes the number of samples which are incorrectly identified as the class j but actually from the class i. The ROC curve illustrates the performance of the classification model through TP and FP. The larger value of the area under the ROC curve (AUC) means the higher TP and the lower FP. In addition, accuracy (i.e., AC = (TP + TN)/(TP + TN + FP + FN)) presents the percentage of correctly classified samples over all samples. Precision (i.e., P = TP/(TP + FP)) and recall (i.e., R = TP/(TP + FN)), respectively, describe the percentage of correctly identified attacks versus all predicted attacks and all actual existing attacks. F-measure (i.e., $F = 2PR/(P + R)$) is the weighted average of precision and recall.

The experimental environments are shown in Table 2. We use Theano [26] to build our neural network model. 80% of the performance of a laptop GPU was used to accelerate calculation speed. The learning rates of pretraining and fine-tuning process were, respectively, 0.001 and 0.1. The minibatch size was 100, and the pretraining epochs were 15.

4.2. Experimental Results and Analysis. We performed three types of classification tasks on the Contagio-CTU-UNB dataset and the CTU-UNB dataset to evaluate the performance of the proposed model. The classification tasks include 6-class classification using the Contagio-CTU-UNB dataset and 2-class and 8-class classification using the CTU-UNB dataset. Specifically, the 6-class classification involves normal data and five kinds of malware traffic data (i.e., botnet, web-based malware, exploit, APT, and scan). The 2-class classification contains normal data and botnet data from the CTU-UNB dataset. The 8-class classification consists of normal data and seven types of botnet data shown in Table 1.

Table 2: Experimental environments.

Name	Configuration
OS	Ubuntu 16.04.1 LTS 64-bit
CPU	Intel Core i5-4200M 2.50 GHz
RAM	8 G
GPU	GeForce GT 740M
Cuda	7.5

Table 3: Accuracy of three kinds of classification tasks.

Metric	6-class			2-class	8-class
	SAE	DBN	DCAE	DCAE	DCAE
Accuracy (%)	96.96	96.94	**98.98**	**99.59**	**98.40**

First, we evaluated the proposed model on three types of the classification tasks. In the 6-class classification task, we also compared our method with other deep learning approaches which have the similar structure and the training process. Furthermore, we evaluated the generalization ability of the proposed model through utilizing the well-trained model of the 2-class classification to detect unknown attacks which are not involved in the training set. Meantime, some important parameters of our model were analyzed.

Table 3 shows the accuracy of three types of the classification tasks. The optimal parameters from experimental tests were used in the DCAE method, as shown in Figure 6 and Table 10. The number of hidden layers of the SAE and the DBN was the same with the DCAE. In the 6-classification task, the DCAE obtained the highest accuracy in comparison with other deep learning methods. Meanwhile, our method performed best and achieved 99.59% accuracy rate in the binary classification. The result of 8-class classification was slightly worse than 6-class classification. The precision, recall,

Table 4: Precision, recall, and F-measure of various deep learning methods.

Traffic type	SAE			DBN			DCAE		
	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)	P (%)	R (%)	F (%)
Normal	96.09	96.60	96.35	96.12	96.12	96.12	98.59	98.64	98.62
Botnet	96.38	97.41	96.90	96.38	97.61	96.99	98.69	99.66	99.17
Web-based Malware	97.87	97.56	97.72	97.72	97.56	97.64	99.12	98.96	99.04
Exploit	96.71	95.28	95.99	96.38	95.84	96.11	98.94	98.73	98.84
APT	95.56	93.48	94.51	97.33	92.39	94.80	99.24	94.93	97.04
Scan	98.50	98.50	98.50	98.58	98.50	98.54	99.84	99.61	99.72
Total	96.96	96.96	96.95	96.95	96.94	96.94	**98.98**	**98.98**	**98.98**

Table 5: Precision, recall, and F-measure of the 8-class classification task.

Traffic type	P (%)	R (%)	F (%)
Normal	99.77	99.57	99.67
Neris	91.63	97.19	94.33
Rbot	97.66	95.74	96.69
Virut	98.76	97.48	98.12
Menti	97.50	90.70	93.98
Sogou	80.00	80.00	80.00
Murlo	96.10	96.37	96.23
NSIS.ay	99.07	98.62	98.84
Average	98.44	98.40	98.41

Table 6: Confusion matrix of the 6-class classification task.

	Normal	Botnet	Web-based malware	Exploit	APT	Scan
Normal	**2033**	10	10	8	0	0
Botnet	3	**2040**	1	1	0	2
Web-based malware	16	4	**1909**	0	0	0
Exploit	8	9	0	**1400**	1	0
APT	2	3	5	4	**262**	0
Scan	0	1	1	2	1	**1264**

and f-measure of 6-class classification are presented in Table 4. The DCAE method also outperformed the compared deep learning methods and achieved the same average value with three metrics after approximation. Table 5 shows the precision, recall, and f-measure of the 8-class classification task. Combining data shown in Table 1, we found that the data size could affect classification results to some degree. Specifically, the class which has fewer data corresponds to the worse performance, which would further affect average values. However, we found that our method still performed well even when there were only few training data. This conclusion can be drawn from Tables 6 and 7.

Table 6 presents the confusion matrix of the 6-class classification task using our method. The leading diagonal shows the number of correctly classified samples of the test set. The botnets and the scans have fewer samples identified incorrectly. Similarly, Table 7 shows the confusion matrix of the 8-class classification task using our method. Though the data size of the menti and the sogou was the smallest, they still achieved relatively good performance. The ROC curves of three types of classification tasks are shown in Figure 5.

The AUC value of binary classification was equal to 1.00, which suggested that our method performed extremely well in the binary classification. Meanwhile, The AUC value of 6-class and 8-class classification was 0.99. It is almost certain that our method produces high true positives and low false alarms.

Additionally, after finishing the 2-class classification task which detected botnet data, the well-trained model was saved in order to evaluate the generalization ability of the proposed model. A new test set containing attack types of the Contagio-CTU-UNB dataset was then constructed to evaluate the generalization of features learned from the CTU-UNB dataset. As shown in Table 8, there are a total of 16000 samples in the new test set which contain two types of traffic data, namely, normal and attack. We chose normal and parts of botnet data from the test set of the CTU-UNB dataset, because the normal data and botnet data of the Contagio-CTU-UNB dataset may be included in the training set of the CTU-UNB dataset. Besides, four types of complex attacks (i.e., web-based malware, exploit, APT, and scan) come from the test set of the Contagio-CTU-UNB dataset. Specifically, we firstly saved the model well trained in the 2-class classification task using the CTU-UNB dataset. The new test set data was then

Table 7: Confusion matrix of the 8-class classification task.

	Normal	Neris	Rbot	Virut	Menti	Sogou	Murlo	NSIS.ay
Normal	**8139**	16	3	10	0	0	6	0
Neris	11	**1521**	7	23	0	0	1	2
Rbot	1	37	**1169**	8	1	1	2	2
Virut	5	74	13	**3672**	0	0	2	1
Menti	0	1	0	0	**39**	0	2	1
Sogou	0	0	0	0	0	**4**	1	0
Murlo	0	3	5	3	0	0	**345**	2
NSIS.ay	2	8	0	2	0	0	0	**855**

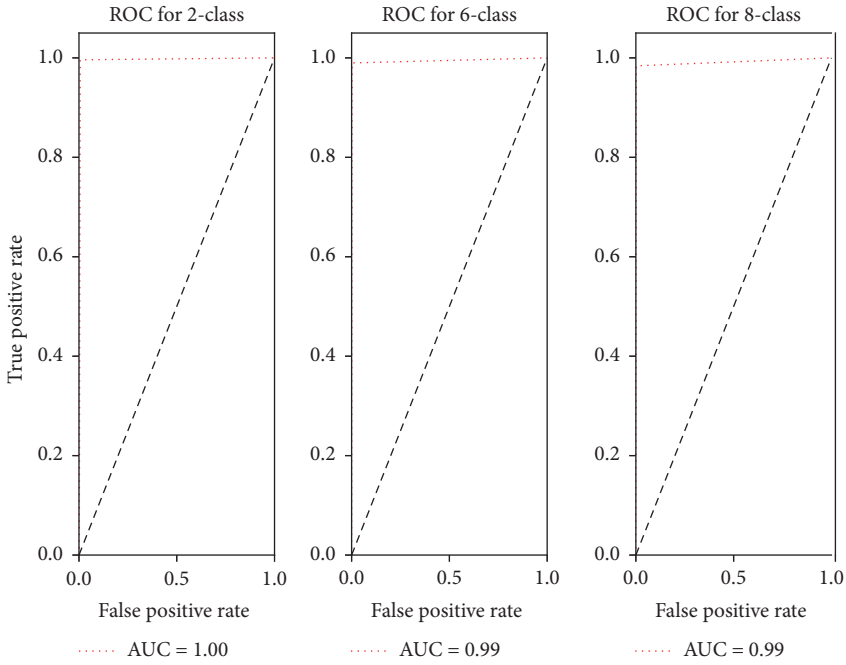

Figure 5: ROC curves for various classification tasks.

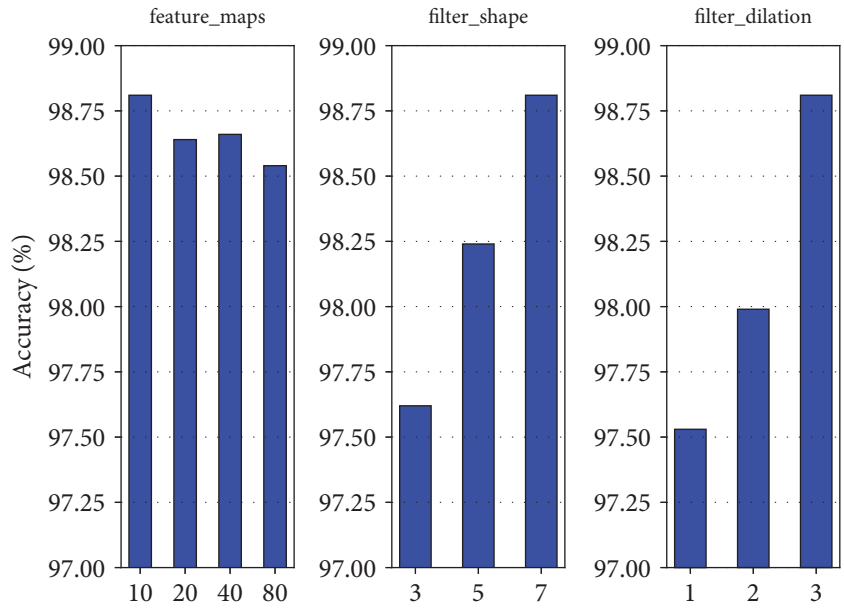

Figure 6: Accuracy comparison of various parameter settings.

Table 8: Sample distribution of new test set for evaluating generalization ability.

Traffic type	Attack					Normal
	Botnet	Web-based malware	Exploit	APT	Scan	
Data size	2934	1929	1418	276	1269	8174
	7826					

Table 9: Precision, recall, and F-measure of the generalization evaluation.

Traffic type	P (%)	R (%)	F (%)
Normal	82.25	99.56	90.08
Attack	99.41	77.56	87.14
Average	90.65	88.80	88.64

evaluated on the saved model. In other words, the proposed model was used to detect some unknown traffic data or various unknown attacks in this generalization evaluation task. The detecting accuracy is 88.8% for the new test set. The precision, recall, and F-measure of the generalization evaluation task are shown in Table 9. The evaluation results suggest that the proposed model could learn some valuable and general features through botnet data to detect unknown attacks.

In the rest of this section, the parameter comparison of experiments used controlling variable method on the 6-class classification task. The controlling variable method means the variable or parameter studied has different value while other parameters are set to optimal values. Figure 6 illustrates accuracy tendency of different parameter settings. The results show that the size of dilation (filter_dilation) and filter (filter_shape) used in the process of convolutional operation has a greater effect on the accuracy rate. Our model got the best performance when the size of dilation and the filter was, respectively, set to 7 and 3. In addition, the number of feature maps (feature_maps) of the convolutional layer has a small effect on the experimental performance. The optimal value of feature maps was set to 10.

Table 10 shows comparative experiments on different numbers of convolutional layers and two types of activation functions used in convolutional autoencoders. The unit number of fully connected layer (i.e., full_units) was set to the same with its input units because we found that the model could get better performance. We chose two types of activation functions for convolutional autoencoders, namely, the sigmoid function (i.e., $f(x) = (1 + e^{-x})^{-1}$) and the ReLU function. These two kinds of activation functions are separately the representatives of saturating nonlinearities and nonsaturating nonlinearities [27]. The cost function corresponding to the sigmoid function was the cross-entropy loss (i.e., $L(\mathbf{x}, \tilde{\mathbf{x}}) = -\sum_{j=1}^{n}[\mathbf{x}_j \log \tilde{\mathbf{x}}_j + (1 - \mathbf{x}_j)\log(1 - \tilde{\mathbf{x}}_j)]$). The experimental results show that the different number of convolutional layers and diverse activation functions do not have a significant effect on the performance of our model. However, they do have a dramatic effect on the run time. The ReLU function is more time-saving compared with the sigmoid function. The number of convolutional layers has a little effect on the run time when the activation function is the ReLU function. Therefore, the ReLU function is more effective than the sigmoid function from the perspective of whole performance.

In addition, we also presented evaluation results on adding a max-pooling layer and various activation functions of the fully connected layer, as shown in Table 11. The parameter settings were the same with the first row of the parameter setting column in Table 10. The experiment on the fully connected layer sets the activation function of convolutional autoencoders to the ReLU function. We added a max-pooling layer before the fully connected layer. We found that the max-pooling operation could not improve the accuracy of our model. But it reduced run time in the scenario of using the sigmoid function and increased run time in the scenario of using the ReLU function. Therefore, it is not suitable and wise to add a max-pooling layer when the activation function of convolutional autoencoders is the ReLU function. Moreover, the run time reached the minimum when the activation function of the fully connected layer was ReLU though the accuracy was not the highest.

We also found that increasing or reducing the length of input vector had little change for the accuracy rate, as shown in Table 12. It may be because the header information and the former parts of traffic payloads are more valuable and useful to identify various attacks. The activation function of the full connected layer was ReLU function. The numeric vector is first transformed into a two-dimensional matrix (such as [10, 20]). We achieved the best performance when the length of input vector was 1000. While the training time of model which has the shorter length of input vectors does not reduce as expected, there could be a tradeoff between training time and accuracy rate. Besides, the number of the units of the fully connected layer also has an optimal range of value. Specifically, the number of the units was better, close to the input of the fully connected lay but not more than the length of sample vectors. That maybe relates to the representation capability of neural networks.

5. Discussion

As stated previously, the purpose of this study was to learn significant features automatically and efficiently from unlabeled raw network traffic data using deep learning techniques. In general, this study shows that the proposed model can achieve high performance by learning feature representations from large volumes of unlabeled training samples. The training samples based on the session are constructed from parts of header and payload information of network packets. We found that the proposed deep learning method obtained quite good results on various classification tasks. These results provide insights into the feature representations learned from raw traffics. It is certain that these feature representations are effective to identify various malicious network traffics and generate low false alarms. The experimental results also show that more layers of convolutional autoencoders fail to

Table 10: Evaluation results on different numbers of convolutional layers and two types of activation functions.

Number of layers	Activation function	Accuracy (%)	Run time (minutes)	Parameter setting
1	Sigmoid	98.83	102.83	filter_dilation = [(3, 3)], filter_shape = [(7, 7)], feature_maps = [10], full_units = 640.
	ReLU	**98.98**	57.09	
2	Sigmoid	98.78	217.25	filter_dilation = [(3, 3), (2, 2)], filter_shape = [(5, 7), (3, 5)], feature_maps = [10, 10], full_units = 960.
	ReLU	98.61	60.64	
3	Sigmoid	98.80	178.26	filter_dilation = [(3, 3), (2, 2), (1, 1)], filter_shape = [(5, 7), (3, 5), (2, 2)], feature_maps = [10, 10, 10], full_units = 690.
	ReLU	98.51	48.65	

Table 11: Evaluation results on adding a max-pooling layer and various activation functions of the fully connected layer.

Layer	Activation function	Accuracy (%)	Run time (minutes)
Max-pooling	Sigmoid	98.42	59.91
	ReLU	97.97	90.41
Fully connected	Tanh	98.97	57.09
	Sigmoid	98.82	101.83
	ReLU	98.62	**8.77**

Table 12: Evaluation results on the different lengths of input vectors.

Length (bytes)	Accuracy (%)	Run time (minutes)	Parameter setting
200 [10, 20]	98.40	19.46	filter_dilation = [(2, 3)], filter_shape = [(3, 7)], full_units = 120.
400 [20, 20]	98.43	37.90	filter_dilation = [(3, 3)], filter_shape = [(4, 7)], full_units = 220.
500 [25, 20]	97.93	13.87	filter_dilation = [(3, 3)], filter_shape = [(5, 7)], full_units = 260.
600 [30, 20]	97.99	19.08	filter_dilation = [(3, 3)], filter_shape = [(6, 7)], full_units = 300.
800 [40, 20]	97.91	16.38	filter_dilation = [(3, 3)], filter_shape = [(7, 7)], full_units = 440.
1000 [50, 20]	**98.62**	**8.77**	filter_dilation = [(3, 3)], filter_shape = [(7, 7)], full_units = 640.
1500 [50, 30]	98.28	52.93	filter_dilation = [(4, 3)], filter_shape = [(10, 9)], full_units = 840.

significantly enhance performance as expected. It is possibly because the the number of hidden units of the first convolutional layer is enough to learn useful feature representations. In addition, diverse activation functions have a great effect on training time. The results suggest that the ReLU function is a good choice for the proposed model to reduce run time. Moreover, an additional max-pooling operation is not necessary for our proposed model compared to traditional convolutional autoencoders.

The limitation of our proposed model is that the training process takes a comparatively long time. However, it can be solved by cross-GPU parallelization technique [27] which is widely used in the deep learning field. In future work, we will implement an online network intrusion detection system in combination with high-performance computing techniques. Additionally, we would try to add missing data or noise and diverse classifiers to enhance the robustness and performance of our system.

6. Conclusion

In this paper, we proposed a novel network intrusion detection model based on dilated convolutional autoencoders. The proposed deep learning method can automatically learn significant feature representations from large volumes of unlabeled raw traffic data. The Contagio-CTU-UNB dataset and the CTU-UNB dataset are created from various malware traffic data. Three kinds of classification tasks are performed to evaluate the performance of the proposed model. We also compared our deep learning method with other similar approaches. The effects of various important hyperparameters are further analyzed. The experimental results show

the superiority of our model by effectively detecting complex attacks from lots of unlabeled data. The remarkable performance we achieved meet requirements of large-scale and real-world network environments by combining high-performance computing techniques.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grants nos. 61379145 and 61105050.

References

[1] Z. Cai, Z. Wang, K. Zheng, and J. Cao, "A Distributed TCAM coprocessor architecture for integrated longest prefix matching, policy filtering, and content filtering," *IEEE Transactions on Computers*, vol. 62, no. 3, pp. 417–427, 2013.

[2] K. Zheng, Z. Cai, X. Zhang, Z. Wang, and B. Yang, "Algorithms to speedup pattern matching for network intrusion detection systems," *Computer Communications*, vol. 62, pp. 47–58, 2015.

[3] Y. Yu, J. Long, F. Liu, and Z. Cai, "Machine learning combining with visualization for intrusion detection: a survey," in *Modeling Decisions for Artificial Intelligence*, vol. 9880 of *Lecture Notes in Comput. Sci.*, pp. 239–249, Springer, Cham, Germany, 2016.

[4] R. Sommer and V. Paxson, "Outside the closed world: on using machine learning for network intrusion detection," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 305–316, IEEE Computer Society, 2010.

[5] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

[6] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: transfer learning from unlabeled data," in *Proceedings of the 24th International Conference on Machine Learning (ICML '07)*, pp. 759–766, ACM, June 2007.

[7] 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.

[8] U. Fiore, F. Palmieri, A. Castiglione, and A. de Santis, "Network anomaly detection with the restricted Boltzmann machine," *Neurocomputing*, vol. 122, pp. 13–23, 2013.

[9] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A Deep Learning Approach for Network Intrusion Detection System," in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, New York, NY, USA, December 2015.

[10] S. Revathi and A. Malathi, *A Detailed Analysis on nsl-kdd Dataset Using Various Machine Learning Techniques for Intrusion Detection*, 2013.

[11] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proceedings of the 2nd IEEE Symposium on Computational Intelligence for Security and Defence Applications*, pp. 1–6, IEEE, July 2009.

[12] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning," *Pattern Recognition*, vol. 58, pp. 121–134, 2016.

[13] Z. Wang, *The Applications of Deep Learning on Traffic Identification*, BlackHat, 2015.

[14] W. Wang, M. Zhu, X. Zeng et al., "Malware traffic classification using convolutional neural network for representation learning," in *Proceedings of the 2017 International Conference on Information Networking (ICOIN)*, pp. 712–717, Da Nang, Vietnam, January 2017.

[15] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–27, 2009.

[16] Y. Fisher and V. Koltun, *Multi-scale context aggregation by dilated convolutions*, 2015, https://arxiv.org/abs/1511.07122.

[17] V. Dumoulin and V. Francesco, *A guide to convolution arithmetic for deep learning*, https://arxiv.org/abs/1603.07285.

[18] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

[19] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proceedings of the 20th Annual Conference on Neural Information Processing Systems (NIPS '06)*, pp. 153–160, Cambridge, Mass, USA, December 2006.

[20] "The ctu-13 dataset," https://stratosphereips.org/category/dataset.html.

[21] "The unb iscx 2012 intrusion detection evaluation dataset," http://www.unb.ca/cic/research/datasets/ids.html.

[22] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357–374, 2012.

[23] "Threatglass by barracuda labs," http://threatglass.com/.

[24] "Contagio malware dump," http://contagiodump.blogspot.kr/2013/08/deepend-research-list-of-malware-pcaps.html.

[25] Y. Yu, J. Long, and Z. Cai, "Session-Based Network Intrusion Detection Using a Deep Learning Architecture," in *Modeling Decisions for Artificial Intelligence*, vol. 10571 of *Lecture Notes in Computer Science*, pp. 144–155, Springer International Publishing, Cham, Germany, 2017.

[26] B. James, B. Olivier, F. Bastien et al., "Theano: A cpu and gpu math compiler in python," in *Proceedings of the 9th Python in Science Conference*, pp. 1–7, 2010.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS '12)*, pp. 1097–1105, Lake Tahoe, Nev, USA, December 2012.