SCIENTIFIC Publishing Limited

Digital Technologies Research and Applications
http://ojs.ukscip.com/index.php/dtra

Article

Network Intrusion Detection with 1D Convolutional Neural Networks

Mohammad Kazim Hooshmand [1,2,*] and ManjaiahDoddaghatta Huchaiah [1]

[1] Department of Computer Science, Mangalore University, Mangalore, India
[2] Department of Computer Science, Kabul Education University, Kabul, Afghanistan
[*] Correspondence: kazimhooshmand@mail.com

Received: 24 May 2022; **Accepted**: 24 June 2022; **Published**: 18 August 2022

Abstract: Computer network assets expose to various cyber threats in today's digital era. Network Anomaly Detection Systems (NADS) play a vital role in protecting digital assets in the purview of network security. Intrusion detection systems data are imbalanced and high dimensioned, affecting models' performance in classifying malicious traffic. This paper uses a denoising autoencoder (DAE) for feature selection to reduce data dimension. To balance the data, the authors use a combined approach of oversampling technique, adaptive synthetic (ADASYN) and a cluster-based under-sampling method using a clustering algorithm, Kmeans. Then, a one-dimensional convolutional neural network (1D-CNN) is used to perform classification. The performance of the proposed model is evaluated on UNSW-NB15 and NSL-KDD datasets. The experimental results show that the model produces a detection rate of 98.79% and 97.23% on UNSW-NB15 for binary classification and multiclass classification, respectively. In the evaluation using NSL-KDD, the model yields a detection rate of 98.52% for binary type classification and 98.16% for multiclass type classification.

Keywords: DAE; ADASYN; feature selection; imbalance processing; NADS; network security; deep learning; CNN

1. Introduction

The pervasive use of digital devices in today's digitally connected world, especially the IoT paradigm in different aspects of lives [1] , brings much information into cyberspace. On the other hand, this overall development and changes cause hidden dangers [2,3] to digital assets. Hence, protecting digital assets is a critical challenge and motivates network security scholars to research the domain. In recent years, many network security incidents have happened on personal and commercial systems with different attack methods [4]; it shows that the number of attacks increased and the forms of attacks increased dramatically. The traditional in-place security tools as a first-line network security defence, such as encryption methods and firewalls, cannot cope with all types of network security dimensions. Therefore, developing an effective tool to detect network attacks is highly needed. NADS is a promising method to identify new intrusive activities and unauthorized access to digital assets.

Moreover, network traffic data is huge, resulting in computational costs for anomaly detection [5]. In the real world, network traffic data are an imbalance that causes a delay in the model's convergence and results in bias prediction [6,7]. Re-sampling techniques are widely employed methods to balance data. Re-sampling includes over-sampling and under-sampling. Each one has its pros and cons. Over-sampling is good for keeping all information but increases the size of data.

In contrast, under-sampling decreases the data size but causes information loss to some extent based on the type and proportion of sampling. The simplest method is random over-sampling (ROS) and under-sampling (RUS).

Some of the other representations are Balance Cascade [8], Synthetic Minority Oversampling Technique (SMOTE) [9] and Adaptive Synthetic (ADASYN) [10]. There are many recommendations for over-sampling in previous works; for example, author [11] has proved that over-sampling does not cause over-fitting, and it is the best way to handle imbalance problems in deep learning. We propose a feature selection technique using DAE to reduce the data dimensionality to solve the problems mentioned earlier. To handle imbalance issues in network traffic data, we use a combined method of oversampling, ADASYN and a cluster-based under-sampling, using the K-means algorithm. The aim of using the combined method is to avoid the data size increase with the help of under-sampling and exploits the advantages of keeping as much as informative samples with the use of over-sampling.

Then, this imbalance processing approach is fused to 1D CNN to perform binary classification and multiclass classification tasks.

The performance of the model is evaluated on NSL-KDD and UNSW-NB15 datasets. Our main contributions in this study are:

a. We present a combined ADASYN and K-means-based clustering method to handle imbalance issues in the network traffic data.

b. We propose a flow-based NADS that employs an integrated imbalance processing method and the 1D CNN model. Our model performs superior to the state-of -the-art in producing detection rates and dramatically reducing false alarm rates. It will be a prominent point for future NADS design and development.

The remainder of this paper consists of several sections. Section 2 briefly describes related work. Next, in Section 3, a description of the datasets is given. The pro-posed method is explained in Section 4. Section 5 details experimental results and analysis, and finally, Section 6 concludes the paper.

2. Related Work

This section briefly summarises some of the most related literature to our work. Authors [12] employed DAE for feature selection and Multilayer Perceptron (MLP) for classification, with an accuracy of 98.80%. The performance of the proposed method was evaluated on the UNSW-NB15 dataset. Authors [13] proposed a network intrusion detection using a conditional variational autoencoder for network anomaly detection in the IoT domain. The proposed method deals with feature re-construction in the case of incomplete data. The authors claimed that the performance was improved and less complex than other unsupervised methods. Authors [14] proposed a Recurrent Neural Network (RNN) based network intrusion detection system. The authors claimed that deep learning-based network intrusion detection achieves better in the case of big data processing. Authors [15] proposed a deep learning-based distributed attack detection in an IoT environment especially using edge devices. The method produced an accuracy of 98.27% on the NSL-KDD dataset. Baig et al. [16] developed a multiclass classification method for network intrusion detection. They used a cascaded artificial neural network which yields an accuracy of 86.40% on the UNSW-NB15. Kwon et al. [17] used a fully connected neural network architecture for NIDS. The performance of the method was tested on the NSL-KDD dataset. The detection rate was reported from 92.9% to 95.3% for different files of the NSL-KDD dataset. The authors [18] proposed an intrusion detection based on Deep Neural Network (DNN). The model performance was evaluated on different datasets for binary and multiclass classification. With the rapid growth of technological advancement, data in cyberspace is getting larger and larger. Therefore, shallow learning with traditional machine learning (ML) relies on a high level of human involvement in data preparation, which may not be suitable in the real-world environment [19,14]. Also, these techniques produce low accuracy [14]. In recent years, deep learning demonstrated success in different real-world problem solving, including cyber-security, due to its capability of automatic feature capturing and correlation in large data- sets [14].

3. Dataset

We use UNSW-NB15 and NSL-KDD datasets for the proposed model performance evaluation.

3.1 UNSW-NB15

The UNSW-NB15 [20,21] was generated by the University of South Wales in 2015. The researchers employed three virtual servers and used a tool called Bro to extract 49-dimensional features, including two labels. This dataset has 2.54 million network traffic samples with nine types of attacks. The number of attack types is more than KDD, and its features are plentiful. UNSW-NB15 has a serious class imbalance. From Table 1, detailed distribution of each class, we can calculate that 87.35% of the entire data is normal traffic, and the remaining 12.65% is all types of attacks. We use the entire dataset for the experiment and divide it into training, testing, and validation at a ratio of 70%, 20%, and 10%,respectively.

3.2 NSL-KDD

NSL-KDD is a refined version of the KDD CUP 99 dataset [22]. The records in the NSL-KDD have been chosen carefully to avoid redundancy issues in the previous version. It contains only a moderate number of records. There are different files with different formats for the NSL-KDD dataset [23]. In this experiment, we used KDDTrain+ and KDDTest+. From Table 1, detailed distribution of each class, we can calculate that 51.88% of the entire data is normal traffic, and the remaining 48.20% is all types of attacks. It is imbalanced but not as much as UNSW-NB15. We divide it into training, testing, and validation at 70%, 20%, and 10%, respectively. Further details on NSL-KDD are available [24]. This is one of the commonly used datasets in NIDS.

4. The Proposed Method

The proposed method mainly consists of data pre-processing, class imbalance handling, classification, and evaluation. Figure 1 illustrates the architecture of the model.

Table 1. The number of instances in each class of UNSW-NB15 and NSL-KDD datasets.

Dataset	Class	Train-Set	Test-Set	Validation-Set	Total
UNSW-NB15	Normal	15,59,255	4,36,755	2,22,751	22, 18,761
	Generic	1,51,430	42,418	21,633	2, 15,481
	Exploits	31,291	8,764	4,470	44,525
	Fuzzers	17,039	4,773	2,434	24,246
	DoS	11,491	3,220	1,642	16,353
	Reconnaissance	9,830	2,753	1,404	13,987
	Analysis	1,881	527	269	2,677
	Backdoors	1,637	458	234	2,329
	Shellcode	1,061	298	152	1,511
	Worms	123	34	17	174
	Total	**17,85,038**	**5,00,000**	**2,55,006**	**25,40,044**
NSL-KDD	Normal	54,150	15,168	7,736	77,054
	DoS	37,517	10,508	5,360	53,385
	Probe	9,893	2,772	1,412	14,077
	R2L	2,634	738	377	3,749
	U2R	177	50	25	252
	Total	**1,04,371**	**29,236**	**14,910**	**1,48,517**

The data pre-processing includes one-hot encoding, data normalization, and feature selection. The second step is combining imbalance processing using ADASYN and the k-means algorithm. The classification consists of 1D-CNN and, finally, the evaluation of the model.

4.1 Data Pre-Processing

First, we dropped unnecessary features such as “srcip”, “ sport”, “dstip”, “dsport”, “stime” , and “ltime” [12] from the UNSW-NB15 dataset before the data pre-processing step. Data pre-processing consists of three main steps: one-hot encoding, data standardization, and feature selection. One-hot encoding is a process of converting nominal features into binary vectors [4]. The goal of performing data standardization is to bring down all the features to a common scale without distorting the differences in the range of the values. Feature selection reduces the number of variables to an optimal set by eliminating redundant or unnecessary variables. UNSW-NB15 and NSL-KDD have three nominal features each. Nominal features of UNSW-NB15 are “proto”, “state”, and “service”,and nominal features of NSL-KDD are “protocol type”, “service”, and “flag”. After applying one-hot encoding, the UNSW-NB15 feature dimension increased to 202 , and the feature dimension of NSL-KDD increased to 121. Similarly, one-hot encoding is applied on the class label of both datasets. Next, we standardize all the remaining features according to Equation (1) and normalize them to Normal Distribution, also called the Gaussian Distribution, with a mean of 0 and a variance of 1.

$$x' = \frac{x-\mu}{\delta} \quad (1)$$

where x is the normalized features, x is the original feature, μ is the mean, and δ is the standard deviation.

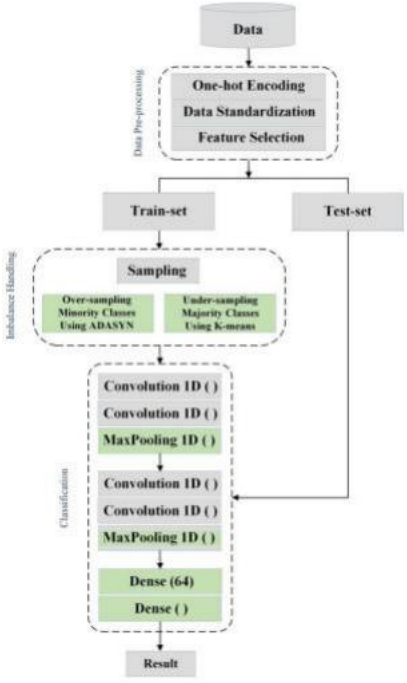

Figure 1. The proposed Method.

The general form of Gaussian Distribution is given in Equation (2).

$$f(x, \mu, \delta) = \frac{1}{\delta\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\delta^2}} \quad (2)$$

where x is the original feature, μ is the mean, and δ is the standard deviation.

Finally, we perform feature selection using DAE on both datasets and select the top 15 features for each. Table 2 shows 15 selected features' names of both datasets.

4.2 Data Imbalance Processing

In Table 1, the number of instances is too small for some classes in the UNSW-NB15. For example, *worms*, *shellcode*, and *backdoors* have few samples, and similarly, the number of samples for *U2R* and *R2L* is smaller in NSL-KDD compared to the other classes. We present a combined over-sampling technique using ADASYN and a cluster-based under-sampling technique using the K-means algorithm to balance the data. Over-sampling alone increases the data size, ultimately increasing the model's computational cost and affecting model accuracy. Under-sampling decreases data size but causes information loss by eliminating some informative transactions. To overcome those shortcomings, we use a combination of both re-sampling techniques. For the minority class, we use the ADASYN technique. ADASYN is an improved version of SMOTE. The key idea behind the use of ADASYN is that it employs a density distribution as a criterion to automatically decide the number of synthetic samples that are required to be generated for each minority example [10]. The way it works is similar to SMOTE, with a minor improvement. It adds some small random values to the points, making them more realistic. So, instead of all the samples being linearly correlated to the parent points, they have some more variance,i.e . they are scattered. Next, in the clustering-based under-sampling technique, we divide the data belonging to each majority class into 10 clusters and randomly choose some portions from each cluster so that the sum of all taken portions has to be equal to the selected threshold the minority class has been over-sampled. Algorithm 1, ADASYN-KM, describes our imbalance processing method.

Table 2. Selected features of UNSW-NB15 and NSL-KDD datasets.

Dataset	Selected features
UNSW-NB15	dtcpb,stcpb, service_-, dmeansz,dload,sload, service_dns, smeansz, trans_depth,sttl, djit,service_ftp-data, ct_ftp, ct_state_ttl,sloss
NSL-KDD	service_http, dst_host_srv_rerror_rate, duration, dst_host_same_srv_rate, protocol_type_udp, same_srv_rate, dst_host_serror_rate, srv_rerror_rate, logged_in, service_telnet, dst_host_srv_serror_rate, service_other, flag_SF, num_root, srv_count,

4.3 Convolutional Neural Networks

Neural networks (NNs) are a subset of ML at the heart of deep learning algorithms. Neural networks compromise from node layers. They contain three main layers, i.e. input layer, one or possibly more hidden layers, and an output layer. The nodes are connected one to another and are associated with weight and a certain threshold. The node is activated if the output value crosses the threshold and then passes the data to the next layer; else, no data is passed to the next network layer. Convolutional neural networks (CNN) are an extension architecture of feed-forward neural networks by having three main layers: convolutional, pooling, and fully connected. The first layer is the convolutional layer, followed by other layers or pooling layers. The final layer of CNN is the fully connected layer.

Algorithm1 ADASYN-KM

Required:

(a) C_n = The total number of classes;
(b) Training set $S = S_i$, $i = 1,2,3,...C_n$;
(c) $|S| = N$; #All samples
(d) cl = 10 #number of clusters

Output:

(a) A balanced training set S;

1: $Avg = int\ (N/C_n)$
2: **for** $i \leftarrow 1$ to C_n **do**
3: **if** $|Si| < Avg$ **then**
4: $S'_i = ADASYN\ (S_i, Avg)$ # Apply ADASYN S'_i, so $|S'|=Avg$
5: **end if**
6: **if** $|S_i| > Avg$ **then**
7: $K_p = K - means\ (S_i, cl)$ #Use K-means algorithm to cluster S_i into cl
Clusters, p = 1, 2, 3...., cl

8: **for** $p \leftarrow$ 1 to *cl* **do**

9: **if** $K_p > Avg/cl$ **then**
10: $K'_p = Resample\ (K_p, Avg/cl)$ #Avg/cl random samples from K_p
11: **else**
12: $K'_p = Avg/cl$ #assign whole cluster
13: **end for**
14: $S'_i = Merge\ (K'_p)$
15: **end if**
16: S'=Merge(S'i)
17: **end for**
18: **return** *S*'

- **Convolutional layers:** The convolutional layer is the main part of CNN because the majority of computation occurs here. The few required components of this layer are input data, a filter, and a feature map.
- **Pooling layer layers**: This layer is responsible for dimensionality reduction and reducing the number of parameters in the input, thus called down-sampling. The pooling operation sweeps the filter through the input as the convolutional layer does, but this filter has no weights. Moreover, the kernel applies an aggregation function intending to populate the outputarray in the pooling operation. The two main types of pooling are Max Pooling and Average Pooling [25,26]. Max pooling selects the maximum value, and Average Pooling calculates the average values within the receptive field. In this work, we use Max Pooling, as shown in Equation (3).

$$f_{max}(x) = max(x_a) \quad (3)$$

where *x* describes the vector of input data with an activation function.

- **Fully connected layers:** The fully connected layer is responsible for the task of classification based on the extracted features through the previous layers and different filters. The fully connected layer usually leverages a *softmax* activation function to classify inputs, while convolutional and pooling layers tend to use *ReLu* functions.

We developed our model based on a six-layer 1D CNN. Figure 1 illustrates the complete step-by-step of our model. The classification part is the network architecture, which shows that a Max-pooling layer follows every two convolutional layers for the first four layers. The dense layers integrate the locally learned features into global features. The first dense has 64 neural units, and the final dense mainly performs the task of classification or prediction. The parameters vary from dataset to dataset based on the number of class labels and type of classification, such as binary and multiclass classification. The data are just mapped into a two-dimensional array as the input to the network.

5. Experimental Results and Analysis

We implemented the proposed CNN-based NADS model in Python and conducted the experiments on a machine with Windows 11 Pro 64 bits operating system. Detailed information on the experimental setup is given in Table 3. The class imbalance process was done on the training set only. The batch size was set to 256, and the epoch was between 100 to 200.

Table 3. Experimental environment.

Parameter	Value
OS	Windows 11 Pro
CPU	Intel® Xeon® W-1250 @3.30 GHz
GPU	NVIDIA Quadro RTX 4000
RAM	32 GB
Programming Language	Python 3.6
Framework	Keras 2.2.4
Backend	Engine Tensorflow

5.1. Evaluation Metrics

The performance of the model is evaluated by Accuracy, Precision, Recall, f-measure, and false alarm rate (false positive rate), which are formulated as:

$$Accuracy = \frac{(TN+TP)}{(TP+FP+TN+FN)} \quad (4)$$

$$Precision = \frac{TP}{TP+FP} \quad (5)$$

$$Recall = \frac{TP}{(TP+FN)} \quad (6)$$

$$F_1 = \frac{2\times(Precision \times Recall)}{(Precision+Recall)} \quad (7)$$

$$FR = \frac{FP}{(FP+TN)} \quad (8)$$

5.2 Classification

The number of convolution kernels and learning rate directly affect classification results in a CNN-based model [27,28]. We performed experiments on several convolution kernels with different learning rates to obtain a better result. This experiment was on UNSW-NB15 for multiclass classification. We use the "nadam" optimizer and "categorical crossentropy" loss

function for the entire experiment. In our model, the number of convolution kernels in the first four layers of 1D CNN is 64 64 256 256. We use Max-pooling two times to under-sample the parameters of the convolution layer. The activation function for the output layer is *softmax*, and for the rest of the layers is *Relu*. To avoid over-fitting, a dropout with a parameter of 0.2 is used after each pooling layer. We tested six different convolution kernels with seven learning rates. Table 4 presents comparative studies of different convolution kernels concerning different sets of learning rates. We observe that the convolution kernels of 64 64 256 256 and learning rate 0.1 outperform in *Recall*, *f1-score*, *Train-loss* and *Test-loss* with the score of 97.23%, 97.64%, 0.30%, and 0.07%, respectively. It scores lower by 0.02% in *Accuracy* with a learning rate of 0.01, and Precision is lower by 0.02% from 64 64 128 128 with a learning rate of 0.1. The FAR is 0.48%, which is higher by 0.8%, with a learning rate of 0.002. Based on this experiment, we can claim that convolution kernels 64 64 256 256 outperform the other number of convolution kernels for most metrics, such as *Accuracy*, *Recall*, *f1-score*, *FAR*, *Train-loss*, and *Test-loss* except for *Precision* and computational time.

To evaluate the effectiveness of our model on different datasets and different types of classifications, we implemented the same experiment on binary classification and multiclass classification on both datasets. Table 5 provides summary results for all the metrics on both datasets. Table 6 presents per class performance of the model on UNSW-NB15 dataset. We can see that the detection rate for the minority classes of shellcode and worms is better concerning the deficient number of samples for the classes.

5.3 Discussion

The experimental results show that by combining an imbalance processing technique and a 1D-CNN-based classifier model, our proposed method significantly improved the detection rate and reduced the FAR. The main reason is that instead of random re-sampling, we employed a combined method of ADASYN over-sampling and a cluster-based under-sampling using the K-means algorithm. We manually set the number of clusters to 10 due to better results from a small pre-test. Cluster-based under-sampling prevents information loss, which may occur by eliminating samples randomly. In this way, it will contribute to good performance.

On the other hand, ADASYN over-sampling generates synthetic points which are more realistic than the randomly generated points. There are many ways of under-sampling, such as K-means algorithm, random under-sampling, Gaussian-based clustering etc. Similarly, there are many over-sampling techniques like random-oversampling, SMOTE etc. We selected ADASYN rather than SMOTE to generate more realistic samples. SMOTE generates samples linearly, while ADASYN works similarly to SMOTE but adds some small fractions to generate more realistic samples. To demonstrate the effectiveness of our model, we compared our results with some of the previous works given in Table 7. We use *Accuracy*, *Precision*, *Recall*, *f-measure*, and *FAR* metrics for the comparison and can see from the table that our model performs much better than the previous works. Reducing the false positive rate (false alarm rate) is one of the key challenges in network anomaly detection, which is significantly dropped by our method.

Table 4. Performance comparison of the proposed model with different numbers of convolution kernels at different learning-rate for multiclass classification on UNSW-NB15 dataset.

Conv. Kernel No.	LR	Acc. %	Precision %	Recall %	F1-Score %	FAR %	Train-Time InSec	Test-Time Insec	Train-Loss	Test-Loss
16_16_32_32	0.1	98.89	98.28	96.79	97.37	0.06	2877	**11**	0.34	0.08
	0.03	98.89	98.32	96.85	97.41	0.07	3143	14	0.34	0.07
	0.01	98.88	98.29	96.82	97.37	0.06	3474	14	0.34	0.08
	0.008	98.89	98.31	96.94	97.43	0.10	3358	13	0.35	0.07
	0.006	98.88	98.30	96.85	97.38	0.08	3432	15	0.35	0.08
	0.004	98.88	98.30	96.92	97.41	0.04	3316	16	0.34	0.07
	0.002	98.88	98.31	96.86	97.40	0.06	3359	17	0.34	0.07
16_16_64_64	0.1	98.96	98.29	96.95	97.47	0.48	**2477**	46	0.33	0.08
	0.03	98.90	98.28	96.77	97.39	0.11	2491	47	0.33	0.07
	0.01	98.90	98.32	96.60	97.28	0.09	2989	56	0.33	0.08
	0.008	98.94	98.31	96.84	97.44	0.22	2746	64	0.33	0.07
	0.006	98.95	98.31	97.03	97.50	0.28	2915	44	0.33	0.07
	0.004	98.89	98.30	96.82	97.40	0.07	2942	64	0.33	0.08
	0.002	98.90	98.31	96.84	97.42	0.08	2948	64	0.33	0.08
32_32_64_64	0.1	98.92	98.37	97.02	97.51	0.16	3226	101	0.32	0.07
	0.03	98.91	98.38	97.07	97.53	0.14	3293	123	0.32	0.07
	0.01	98.91	98.38	96.60	97.33	0.12	3418	125	0.32	0.09
	0.008	98.92	98.37	97.02	97.52	0.16	3351	121	0.32	0.07
	0.006	98.91	98.34	96.62	97.31	0.08	3287	124	0.32	0.07
	0.004	98.92	98.37	97.05	97.53	0.15	2956	97	0.32	0.07
	0.002	98.91	98.37	97.02	97.51	0.13	3292	127	0.32	0.07
32_32_128_128	0.1	98.94	98.37	97.08	97.55	0.26	4087	215	0.31	0.07
	0.03	98.99	98.37	97.03	97.55	0.41	4183	230	0.31	0.07
	0.01	98.91	98.37	96.97	97.49	0.14	3801	196	0.31	0.07
	0.008	98.92	98.37	97.02	97.52	0.13	4214	224	0.31	0.07
	0.006	98.92	98.39	96.99	97.51	0.10	4217	225	0.31	0.07
	0.004	98.93	98.40	96.94	97.51	0.13	4135	218	0.31	0.07
	0.002	98.93	98.37	96.86	97.45	0.17	4172	229	0.31	0.07
64_64_128_128	0.1	98.96	**98.40**	97.14	97.59	0.29	5238	326	0.30	0.07
	0.03	98.98	98.39	97.10	97.58	0.29	4875	275	0.30	0.07
	0.01	98.99	98.38	97.19	97.61	0.39	5482	348	0.30	0.07
	0.008	98.99	98.39	97.12	97.60	0.35	5438	415	0.30	0.07
	0.006	98.98	98.38	97.17	97.60	0.38	5628	421	0.30	0.07
	0.004	98.97	98.38	97.16	97.59	0.29	5373	364	0.30	0.07
	0.002	98.96	98.39	97.15	97.59	0.29	5497	375	0.30	0.07
64_64_256_256	0.1	99.01	98.38	**97.23**	**97.64**	0.48	6030	339	**0.30**	**0.07**
	0.03	98.99	98.39	97.14	97.60	0.37	6673	389	0.30	0.07
	0.01	**99.03**	98.38	97.03	97.57	0.48	6949	414	0.30	0.07
	0.008	99.02	98.37	97.14	97.61	0.49	7373	439	0.30	0.07
	0.006	98.99	98.37	97.17	97.61	0.40	7388	440	0.30	0.07
	0.004	99.00	98.38	97.09	97.59	0.42	7621	454	0.30	0.07
	0.002	99.00	98.38	97.16	97.61	**0.40**	7846	467	0.30	0.07

Table 5. Performance of the proposed model in binary/multiclass classification on UNSWNB15 and NSL-KDDdatasets.

Dataset	**UNSW-NB15**		**NSL-KDD**	
Classification	**Binary**	**Multi-class**	**Binary**	**Multi-class**
Accuracy %	98.79	99.01	98.52	98.97
Precision %	98.90	98.38	98.52	98.35
Recall %	98.79	97.23	98.52	98.16
F1-score %	98.81	97.64	98.52	98.22
FAR %	0.18	0.48	1.48	0.48
Train-time in sec.	8049.91	6029.59	641.41	626.41
Test-time in sec.	388.31	339.09	15.67	14.90
Train loss	0.03	0.30	0.09	0.17
Test_loss	0.02	0.07	0.05	0.07

Table 6. Performance evaluation of the proposed model in multiclass classification on the UNSW-NB15 dataset.

Class	**Accuracy %**	**Precision %**	**Recall %**	**F1-score %**	**Support**
Normal	98.92	99.92	98.84	99.38	4,36,755
Generic	99.83	99.84	98.21	99.02	42,418
Exploits	98.97	84.07	50.54	63.13	8,764
Fuzzers	98.90	45.22	69.70	54.85	4,773
DoS	98.90	33.48	71.74	45.65	3,220
Reconnaissance	99.82	85.76	80.28	82.93	2,753
Analysis	99.69	11.34	28.46	16.22	527
Backdoors	99.77	10.82	20.74	14.22	458
Shellcode	99.68	13.13	78.52	22.50	298
Worms	99.97	16.13	73.53	26.46	34
Overall/avg.	**99.01**	**98.38**	**97.23**	**97.64**	**5,00,000**

Table 7. Comparison results based on UNSW-NB15 and NSL-KDD datasets for multiclass classification with previous studies.

Dataset %	**Classification**	**Model**	**Acc. %**	**Precision%**	**DR %**	**F1-score %**	**FAR**
UNSW-NB15 19.01	Multiclass	CSCADE-ANN [16]	86.40	86.74	93.38	89.94	-
		ICVAE-DNN [29]	89.08	86.05	95.68	90.61	
		In this work	**99.01**	**98.38**	**97.23**	**97.64**	**0.48**
NSL-KDD	Multiclass	SCDNN [30]	72.64	-	57.48	-	-
		RNN-IDS [14]	81.29	-	-	-	-
		ID-CVAE [13]	80.10	81.59	80.10	79.08	-
		Gaussian–Bernoulli RBM [31]	73.23	62.33	95.09	75.30	-
		ICVAE-DNN [29]	85.97	97.39	77.43	86.27	2.74
		In this work	**98.97**	**98.35**	**98.16**	**98.22**	**0.48**

6. Conclusions

We proposed a combined approach to process class imbalance to overcome the class imbalance issue in network traffic data. The technique combines ADASYN over-sampling and clustering-based under-sampling using the k-means algorithm. We develop a 1D CNN-based network anomaly detection with 64 64 256 256 number of convolutional kernels, a learning rate of 0.1, a softmax activation function for the output layer, Relu activation function for the layers other than the output layer, and Max-pooling followed by a dropout. We evaluated our model on two commonly used datasets, UNSW-NB15 and NSL-KDD. The binary and multiclass classification was conducted on both datasets. A comparative study with six different convolution kernels and seven learning rates was conducted to show how our model performs vs the other convolution kernels. The experimental results show that the model produces a detection rate of 98.79% and 97. 23% on UNSW-NB15 for binary classification and multiclass classification, respectively, which is the highest among all tested scenarios. In the evaluation using NSL-KDD, the model yields a detection rate of 98.52% for binary type classification and 98.16% for multiclass type classification. We compared our method with some previous work on multiclass classification. Our model performs superior to state-of-the-art models and points to a promising direction for future network anomaly detection with large-scale and imbalanced datasets. We plan to explore more imbalance processing methods on a distributed platform to improve detection performance and reduce time in our future research.

Conflict of Interest

There is no conflict of interest.

References

1. Tahaei, H., Afifi, F., Asemi, A., et al., 2020. The rise of traffic classification in IoT networks: A survey. Journal of Network and Computer Applications. 154, 102538.
2. Bernardi, M.L., Cimitile, M., Distante, D., et al., 2019. Dynamic malware detection and phylogeny analysis using process mining. International Journal of Information Security. 18(3), 257–284.
3. Giovanni, A., Bernardi, M.L., Cimitile, M., et al., 2018 . A fuzzy clustering-based approach to study malware phylogeny. 2018 IEEE International Con- ference on Fuzzy Systems (FUZZ-IEEE). pp. 1–8.
4. Liu, Ch., Gu, Zh.J., Wang, J.L., 2021. A hybrid intrusion detection system based on scalable k-means+ random forest and deep learning. IEEE Access. 9, 75729–75740.
5. El-Khatib, K., 2009. Impact of feature reduction on the efficiency of wireless intrusion detection systems. IEEE Transactions on Parallel and Distributed Systems. 21(8), 1143–1149.
6. Zhang, H., Li, J.L., Liu, X.M., et al., 2021. Multi-dimensional feature fusion and stacking ensemble mechanism for network intrusion detection. Future Generation Computer Systems. 122, 130–143.
7. Kumar, G., 2020. An improved ensemble approach for effective intrusion detection. The Journal of Supercomputing. 76(1), 275–291.
8. Liu, X.Y., Wu, J.X., Zhou, Zh.H., 2008. Exploratory undersampling for class-imbalance learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics). 39(2), 539–550.
9. Chawla, N.V., Bowyer, K.W., Hall, L.O., et al., 2002. Smote: synthetic minority over-sampling technique. Journal of Artificial Intelligence Research. 16, 321–357.
10. He, H.B., Bai, Y., Garcia, E.A., et al., 2008. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. 2008 IEEE International Joint Conference on Neural Networks (IEEE world congress on computational intelligence). pp. 1322–1328.
11. Mateusz, B., Atsuto, M., Mazurowski, M.A., 2018. A systematic study of the class imbalance problem in convolutional neural networks. Neural Networks. 106, 249–259.
12. Zhang, H.P., Wu, C.Q., Gao, Sh., et al., 2018. An effective deep learning based scheme for network intrusion detection. 2018 24th International Conference on Pattern Recognition (ICPR). pp. 682–687.
13. Manuel, L.M., Belen, C., Antonio, S.E., et al., 2017. Conditional variational autoencoder for prediction and feature recovery applied to intrusion detection in IoT. Sensors. 17(9), 1967.
14. Yin, Ch.L., Zhu, Y.F., Fei, J.L., et al., 2017. A deep learning approach for intrusion detection using recur- rent neural networks. IEEE Access. 5, 21954–21961.
15. Diro, A.A., Chilamkurti, N., 2018. Distributed attack detection scheme using deep learning approach for internet of things. Future Generation Computer Sys- tems. 82, 761–768.
16. Baig, M.M., Awais, M.M., El-Alfy, E.M., 2017. A multiclass cascade of artificial neural network for network intrusion detection. Journal of Intelligent & Fuzzy Systems. 32(4), 2875–2883.
17. Kwon, D., Kim, H., Kim, J., et al., 2019. A survey of deep learning-based network anomaly detection. Cluster Computing. 22(1), 949–961.
18. Vinayakumar, R. Alazab, M., Soman, K.P., et al, 2019. Deep learning approach for intelligent intrusion detection system. IEEE Access. 7, 41525–41550
19. Pasupa, K., Sunhem, W., 2016. A comparison between shallow and deep architecture classifiers on small dataset. 2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE). pp. 1–6.
20. The UNSW-NB15 data set description.
21. Moustafa, N., Slay, J., 2015. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). 2015 military communications and information systems conference (MilCIS). pp. 1–6.
22. Hinton, G.E., Osindero, S., Teh, Y.W., 2006. A fast learning algorithm for deep belief nets. Neural Computation. 18(7), 1527–1554.
23. Laqtib, S., Yassini, K.E. , Hasnaoui, M.L , 2019. Evaluation of deep learning approaches for intrusion detection system in manet. The Proceedings of the Third

International Conference on Smart City Appli- cations. pp. 986–998.
24. TheNSL-KDD data set description.
25. O'Shea, K., Nash, R., 2015. An introduction to convolutional neural networks. Computer Science.
26. Wu, H.B., Gu, X.D., 2015. Max-pooling dropout for regularization of convolutional neural networks. International Conference on Neural Information Processing. pp. 46–54.
27. Maitra, S., Ojha, R.K., Ghosh, K., 2018. Impact of convolutional neural network input parameters on classification performance. 2018 4th International Conference for Convergence in Technology (I2CT). pp. 1–5.
28. Wen, L., Gao, L., Li, X.Y., et al., 2021. Convolutional neural network with automatic learning rate scheduler for fault classification. IEEE Transactions on Instrumentation and Measurement. 70, 1–12.
29. Yang, Y.Q., Zheng, K.F., Wu, Ch.H., et al., 2019. Improving the classification effectiveness of intrusion detection by using improved conditional variation- al autoencoder and deep neural network. Sensors. 19(11), 2528.
30. Ma, T., Wang, F., Cheng, J.J., et al., 2016. A hybrid spectral clustering and deep neural network ensemble algorithm for intrusion detection in sensor networks. Sensors. 16(10), 1701.
31. Imamverdiyev, Y., Abdullayeva, F., 2018. Deep learning method for denial of service attack detection based on restricted boltzmann machine. Big Data. 6(2), 159–169.