Cybersecurity

RESEARCH

Open Access

Practical autoencoder based anomaly detection by using vector reconstruction error

Hasan Torabi[1], Seyedeh Leili Mirtaheri[1*] and Sergio Greco[2]

Abstract

Nowadays, cloud computing provides easy access to a set of variable and configurable computing resources based on user demand through the network. Cloud computing services are available through common internet protocols and network standards. In addition to the unique benefits of cloud computing, insecure communication and attacks on cloud networks cannot be ignored. There are several techniques for dealing with network attacks. To this end, network anomaly detection systems are widely used as an effective countermeasure against network anomalies. The anomaly-based approach generally learns normal traffic patterns in various ways and identifies patterns of anomalies. Network anomaly detection systems have gained much attention in intelligently monitoring network traffic using machine learning methods. This paper presents an efficient model based on autoencoders for anomaly detection in cloud computing networks. The autoencoder learns a basic representation of the normal data and its reconstruction with minimum error. Therefore, the reconstruction error is used as an anomaly or classification metric. In addition, to detecting anomaly data from normal data, the classification of anomaly types has also been investigated. We have proposed a new approach by examining an autoencoder's anomaly detection method based on data reconstruction error. Unlike the existing autoencoder-based anomaly detection techniques that consider the reconstruction error of all input features as a single value, we assume that the reconstruction error is a vector. This enables our model to use the reconstruction error of every input feature as an anomaly or classification metric. We further propose a multi-class classification structure to classify the anomalies. We use the CIDDS-001 dataset as a commonly accepted dataset in the literature. Our evaluations show that the performance of the proposed method has improved considerably compared to the existing ones in terms of accuracy, recall, false-positive rate, and F1-score metrics.

Keywords: Cloud, Practical, Anomaly detection, Autoencoder, Reconstruction error, Machine learning

Introduction

In recent years, the concept of cloud computing has emerged as one of the most important computing paradigms. Cloud computing has revolutionized information technology and access to processing and computing resources [1, 2]. With cloud computing, individuals and organizations can access a shared network of managed and scalable IT resources such as servers, storage, and applications on-demand [3]. It makes platforms and software available to users as a service. With many advances in recent years, it has become a platform for various users' technologies. These services are based on large networks such as the internet [4] and a model for providing and consuming resources and services that provides access to resources flexible and scalable manner based on user demand and in real-time. By using cloud computing, data and applications can be stored in the cloud and accessed anytime, anywhere through all internet access tools [5]. Cloud computing is recognized as a dynamic provider of computing services over the internet [6].

Considering many advantages of cloud computing, various challenges have arisen. One of the most important challenges in cloud computing is the security

*Correspondence: mirtaheri@khu.ac.ir; leili.mirtaheri@dimes.unical.it

[1] Department of Electrical and Computer Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran
Full list of author information is available at the end of the article

considerations [7]. Due to the increasing use of cloud computing, various cyber-attacks are carried out for different purposes in the cloud [8]. Open and distributed structures in cloud computing are an important target for cyber-attacks by attackers. As cloud services are provided over the internet, communication and information security are still key issues [9].

On the other hand, security attacks, and incidents can significantly impact cloud customers [10]. When cloud services go offline or software and websites crash, it can cause major problems for users who rely on them for day-to-day operations. It means losing revenue, losing customers, and losing reputation for businesses. Because the cloud environment has shared network resources between consumers, the risk increases, and steps must be taken to improve security. Many approaches and methods for security monitoring in cloud computing networks have been introduced [11]. Anomaly detection using machine learning algorithms is widely used and helps detect suspicious or abnormal data that differs significantly from the normal data [12, 13].

This research mainly focuses on anomaly detection and classification of anomalies in cloud network data using autoencoders. The anomaly detection approaches generally learn normal patterns and detect anomaly samples whose patterns significantly deviate from the normal ones. We extend this method for classifying all class data in the dataset. For instance, a model learns class patterns and detects other class samples whose patterns significantly deviate from that class data ones. The idea of using autoencoders for anomaly detection is already presented in the literature [14]. An autoencoder is a neural network consisting of an encoder and a decoder trained to learn reconstructions close to the original input. The difference between the original input and the reconstruction output in the autoencoder is called the reconstruction error.

An autoencoder-based anomaly detection system trained with only normal traffic data is expected to recover any given input as close as possible to the learned normal patterns. Therefore, we can classify an input instance as an attack if its reconstruction error is larger than a predefined threshold; otherwise, we can classify the input instance as normal. In this method, an autoencoder-based anomaly detection system can detect unknown types of attacks when their patterns deviate from the learned normal patterns. The existing methods have some issues, and there is room for further improvements. One important issue in existing methods is treating the reconstruction error as a single value. In existing methods, the reconstruction error of all input vector elements or features is summed up in one value. Since the threshold selection is highly dependent on reconstruction error, the classification based on this threshold is far from ideal.

In this paper, we propose a new approach for anomaly detection based on autoencoders. We assume vector instead of single value and consider reconstruction error and threshold for every feature. In fact, instead of using the summation of reconstruction error in one value, we create a vector of the reconstruction error. This approach improves the performance of anomaly detection and classification. To evaluate the proposed method, we adopt the CIDDS-001 dataset [15]. In addition, we investigate the effectiveness of a hierarchical multi-classification on the performance of the proposed method.

In summary, the contributions of this paper are listed as follows:

- We propose a new method to use autoencoders in anomaly detection of cloud networks. Unlike the existing methods, we create a vector reconstruction error for every feature. This leads to a threshold vector in anomaly detection that further improves the performance compared to existing methods. Our presented autoencoder network has only one hidden layer and therefore, is more efficient compared with existing autoencoder based anomaly detection methods in cloud.
- We propose a multi-class classifier with a hierarchical structure to classify all classes of data and improve the performance of one class classifier.
- We perform an extensive evaluation of the proposed methods. We adopt the commonly accepted CIDDS-001 dataset and compare the proposed methods with some of the existing ones in terms of accuracy, recall, false-positive rate, and F1-score metrics. The results show considerable improvements. We have made our code publicly available on GitHub. https://github.com/Hasan-Torabi/Anomaly-detection-in-cloud-computing-networks-by-using-autoencoders.

The remainder of this paper is organized as follows: "Literature review" section reviews the literature and discusses related works. "System model" section presents the system model, and the proposed methods are presented in "Proposed model" section. The evaluation setup, experimental results, and comparisons are presented in "Evaluation results and discussion" section, and finally, "Conclusions" section concludes the paper.

Literature review

Cloud computing is a leading technology that has changed how traditional services are provided and has affected the entire IT sector. Security issues are one of the most important challenges of cloud networks. Many algorithms have been proposed over time to overcome security and privacy issues. Facing new cyber issues

or improving the performance of the existing systems, researchers continue to present new methods. One of the promising solutions is to use anomaly detection systems in cloud networks to prevent system failure, attacks, and intrusion [16]. In this section, we present some of the existing and related methods that use anomaly detection in cloud networks.

The authors in [17] use a deep neural network (DNN) to classify the types of attacks on cloud and internet of things (IoT) networks. They use the cross-validation and repeated cross-validation on the CIDDS-001 dataset to evaluate their proposed method. Authors also apply these methods to different randomly selected sets from the dataset. To find the optimal parameter of DNN, the authors use the grid search method for hyper-parameters optimization.

Long-Short Term Memory (LSTM) for multi-class intrusion detection is also used for the anomaly-based network intrusion detection system in [18]. The authors use the CIDDS-001 dataset and Accuracy metric for performance evaluations. The LSTM model compared with other methods such as support vector machines (SVM), Naïve Bayes (NB), and Multi-Layer Perceptron (MLP). The authors show that the LSTM outcomes the mentioned existing methods in terms of accuracy. However, sequence size and some hyper-parameter values are not mentioned in detail, and the achieved accuracy of LSTM is 85.5%.

Authors in [19] propose a comparative analysis of benchmark datasets NSL-KDD and CIDDS-001 using multiple machines and deep learning classifiers. The authors use the hybrid feature selection and ranking methods. They consider six classification algorithms, including SVM, Naïve Bayes, k-nearest neighbors (KNN), Neural Networks (NN), DNN, and denoising autoencoder (DAE), to measure the performance and accuracy of algorithms in used datasets. The evaluation results show that kNN, SVM, NN, and DNN algorithms achieve higher performance on the NSLKDD dataset, and DAE and Naïve Bayes algorithms achieved high performance on the CIDDS-001 dataset.

In [20], the authors use a deep autoencoder as an intrusion detection system that works based on anomaly detection. Their proposed system trains only with normal data, and the pattern of normal data is learned only by using the characteristics of normal behavior. Every input data is classified according to the reconstruction error threshold to normal or anomaly. The grid search technique is used to tune the autoencoders' parameters. They use the CIDDS-001 dataset to evaluate the performance of their proposed method. This method only performs binary classification, which classifies normal and anomaly data.

An anomaly detection system in a cloud network that uses supervised machine learning is proposed in [12]. The authors use two one-class classifiers instead of multi-class classifiers. They used One-Class Support Vector Machine (OCSVM) and Autoencoder algorithms trained to detect anomalies. The proposed algorithms are tested on the yahoo and the UNSW-NB15 dataset. The experimental results show that the autoencoder outperforms the OCSVM in anomaly detection scenarios.

In [21], a hybrid deep learning-based model for anomaly detection in cloud computing environments is proposed. The model takes advantage of multi-objective optimization and deep learning for feature extraction and anomaly detection on network traffic streams. The authors use the grey wolf optimization (GWO) algorithm for multi-objective feature extraction and a convolutional neural network (CNN) for anomaly classification. Their suggested model contains two phases, feature selection, and classification. Feature selection is performed using improved GWO (ImGWO), and in the second phase, an improved CNN (ImCNN) is applied for classification. The authors evaluate the proposed model's efficacy using DARPA'98 and KDD'99 and synthetic datasets. They show that their proposed ImGWO and ImCNN based model performs better results compared with standard GWO and CNN.

Authors in [22] use deep learning for intrusion detection in IoT networks. The authors develop feed-forward neural network models by proposing an intelligent intrusion detection approach. Transfer learning encodes high-dimensional features applied for multi-class classification for building a binary classifier. A detection model is also trained to detect four attacks in IoT networks. The efficacy of the proposed model is evaluated on a dataset comprising realistic network trac.

A software-defined network-based anomaly detection system (SDN-ADS) is proposed for cloud computing, and edge computing-based networks in [16]. Cloud computing and edge computing networks have suffered from security concerns due to different malicious activities and security attacks. These malicious activities lead to link failure and wrong forwarding decisions and divert the paths. The proposed SDN anomaly detection system-based network in [16] is divided into different layers based on north and south application programming interfaces (APIs) and malicious switches. A Trusted Authority for Edge computing model is used to ensure the trust in edge devices; when the trust is established, then all communication can be performed through local certificates.

Authors present an analysis of the CIDDS-001 dataset for anomaly-based network intrusion detection systems from the machine learning perspective [23]. They use

k-nearest neighbor (KNN) classication and k-means clustering techniques to measure the complexity in terms of prominent metrics. Based on the presented evaluation results, the authors show that both k-nearest neighbor classication and k-means clustering perform well over the CIDDS-001 dataset in terms of used prominent metrics.

An artificial neural network (ANN) model is used for anomaly detection in [24]. The authors use regularization to improve generalization by reducing model complexity. They propose a new regularization technique for anomaly detection based on the standard deviation of the weight matrix. They show that their proposed regularization algorithm is capable of identifying good patterns in data and classifying them efficiently.

Authors in [25] perform anomaly detection based on the network flow features. They evaluate the detection capabilities of both known and unknown attacks with autoencoder and variational autoencoder deep learning methods with a one-class support vector machine. Their presented models are created only by using normal ow-based data. The proposed autoencoder and variational autoencoder in [25] have two encoding and two decoding layers, with the bottleneck layer having 64 neurons. The encoder involves an experiment on the CICDS2017 dataset, extraction of the stream-based features, and a calculation of the region of convergence (ROC) curve and the area under the curve (AUC) value. The results show that the AUC value obtained by the variational autoencoder is better than that of the autoencoder and single-class support vector machine. Still, it is not easy to determine an appropriate threshold that provides high detection accuracy or a low false alarm rate.

Authors in [26] propose a cyber-attack detection system for networks. In this research, a sequential approach for intrusion detection by using time-based transformations in the algorithm's input data is proposed. An experiment is conducted with random forest, multi-layer perceptron, and long-short term memory to understand the best performance by comparing single-flow and multi-flow detection approaches in the CIDDS-001 dataset.

In [27], the authors suggest a method for assessing network security posture based on stacked autoencoder networks and backpropagation neural networks, which can further reduce model complexity. First, the suggested approach extracts and normalises network domain indicator data. After that, a stacked autoencoder network is utilised to reduce dimension and extract features. The network security situation value is then computed using a backpropagation neural network technique, which can quantitatively analyse the network domain security situation. They evaluated the performance of their proposed technique using the CIDDS-001 dataset. Finally, they demonstrated that the suggested method can accurately assess the security condition of a network area through a series of comparative tests. Furthermore, this method can reduce the dimensionality of input data while keeping valuable data properties, which can save storage overhead and computer resources while improving assessment efficiency.

In reviewed research, many approaches and methods are used for anomaly detection. In all autoencoder-based existing research, authors use a single value for reconstruction error, causing the threshold to be a single value. In this paper, we create a vector of reconstruction error for every feature that further improves the performance of anomaly detection in the networks.

System model

Autoencoder (AE) [28, 29] is a specific type of feed-forward neural network where the input is the same output. The output layer has the same dimension as the input layer, as shown in Fig. 1. These networks use an unsupervised approach for training input vectors to reconstruct as output vectors [30]. AEs are made up of an encoder and a decoder. The architecture of a sample autoencoder is presented in Fig. 1.

The encoder transforms the input vector X to a hidden representation H as presented in Eq. 1.

$$H = \sigma(W_{xh}X + b_{xh}) \tag{1}$$

where σ is an activation function such as a sigmoid function or rectified linear unit, W is a weight matrix, and b is a bias vector. The transformation operation is applied to hidden representation H to reconstruct the initial input space using a decoder.

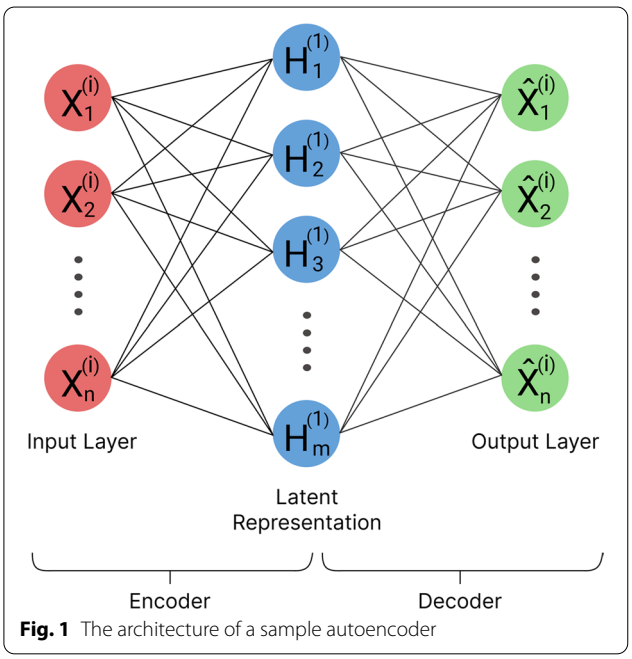

Fig. 1 The architecture of a sample autoencoder

$$\hat{X} = \sigma(W_{h\hat{x}}h + b_{h\hat{x}}) \quad (2)$$

The difference between the reconstructed vector $\hat{X}$ and the original input vector X yields the reconstruction error (RE), r as follows:

$$r = \|X - \hat{X}\| \quad (3)$$

The AE is trained to minimize the r with an unsupervised training approach [25]. The flow chart of AE training is illustrated in Fig. 2.

The RE is the criterion by which anomalies are detected. An AE is trained to minimize this reconstruction error. Therefore, the AE learns the relationships between the features of the input set. If we feed a trained AE with the data (not seen during the training phase) that resembles the data used for the training, AE should reproduce the input with good accuracy in the output. If this is not the case, the AE will be unable to reconstruct the input correctly, resulting in a larger error. This large error enables us to detect anomalies by observing the magnitude of the RE. It means that the RE is used as the anomaly score in AE-based anomaly detection. If an input enters in AE and creates high RE, it is assumed to be an anomaly. The training of AE is done through the normal data. The trained AE model will successfully reconstruct normal input data with very low RE. However, it will fail to do the same with anomaly data it has never seen before. Figure 3 shows the flow chart of the AE-based anomaly detection algorithm [25].

Proposed model

The existing AE-based anomaly detection methods consider the reconstruction error in one value as is shown in Fig. 4.

In these methods, the reconstruction errors of all inputs (or data) features add up together. Therefore, we can not detect the error value of each feature. It mean that the RE of different data might be equal while one data that has error in m features and another data that has error in n features.

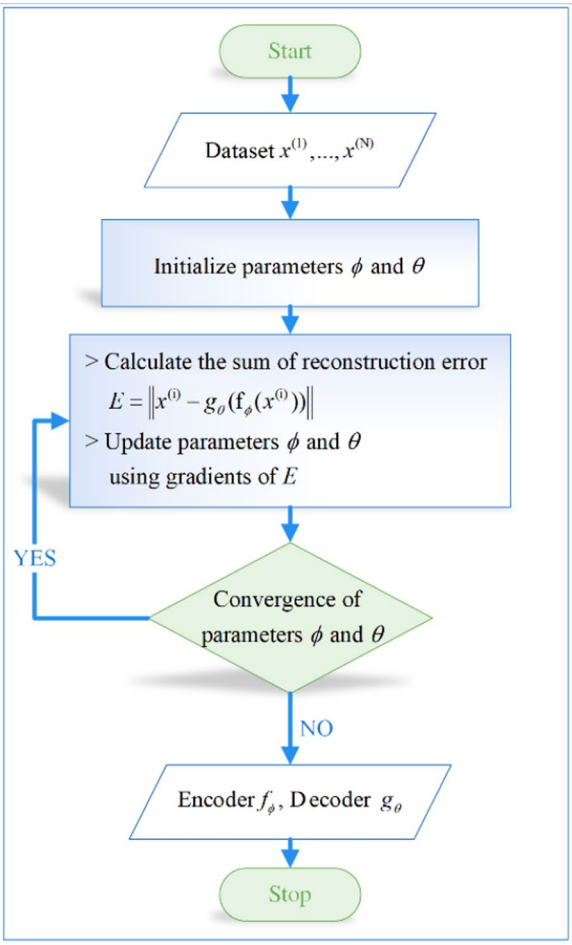

Fig. 2 The flow chart of the autoencoder training algorithms

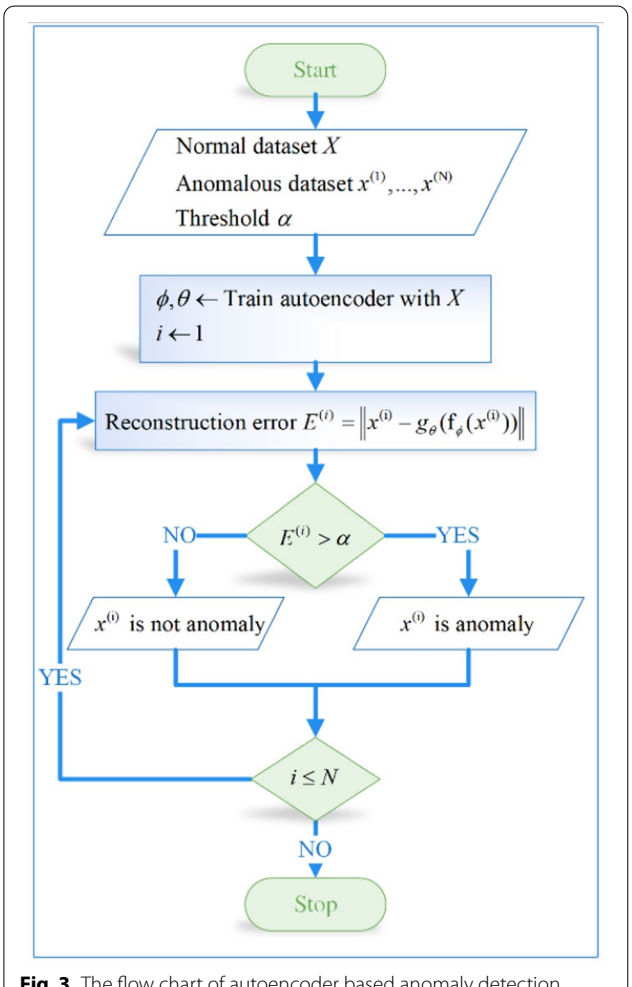

Fig. 3 The flow chart of autoencoder based anomaly detection algorithms

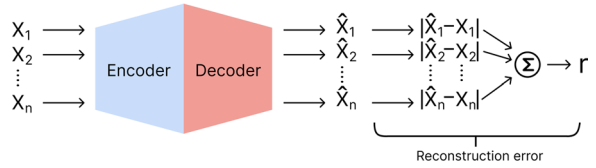

Fig. 4 The procedure of calculation RE in existing AE-based anomaly detection methods

In evaluating this method, the RE of normal and anomaly data overlaps and intersections. This is because the RE summarizes all the features of RE into one value and a threshold with one value specifies the boundary between normal data and anomaly data. For example, we feed anomaly data into trained AE for anomaly detection. If this anomaly data in k features cannot be reconstructed well at the AE output, significant errors are generated in the k features. However, very few errors may be calculated in the other features. Accordingly, when the RE is calculated (only one value), the RE value may be less than the threshold. This causes a false classification due to the inherent weakness of existing methods. To overcome this issue, having a single value of RE is not desirable.

In this paper, we adopt the innovation in calculating the RE and determining the threshold value by considering the RE and the threshold as a vector. In this case, the RE of each output data from AE is equal to the length of input data features, in each of which the RE of that corresponding feature is inserted. RE of each feature is equal to the absolute value of the difference between input and output values of the feature from the AE. First, by training the AE with only normal data, we select the threshold that the RE in each feature is less than that RE value for all normal data. Therefore, for each feature of data, we select a unique threshold. On the other hand, we ensure that the RE of all the features of all normal data is less than these threshold values. In this case, if the RE of one/more feature(s) exceeds the corresponding threshold of that feature, we can classify that data as an anomaly. After training the AE to select the threshold of each feature, we consider the RE according to Eq. 4 as shown in Fig. 5.

$$r = (\|X_1 - \hat{X}_1\|, \|X_2 - \hat{X}_2\|, \ldots, \|X_n - \hat{X}_n\|) \tag{4}$$

where n is a number of features.

The threshold value for each AE is given according to Eqs. 5 and 6.

$$th = (th(X_1), th(X_1), \ldots, th(X_n)) \tag{5}$$

$$th(X_k) = Max(\|X_k^1 - \hat{X}_k^1\|, \|X_k^2 - \hat{X}_k^2\|, \ldots, \|X_k^m - \hat{X}_k^m\|) \tag{6}$$

where m is the number of data samples, this process is illustrated in more detail in the pseudo-code of Algorithm 1. The threshold value of each feature is equal to the maximum error of that feature among the REs of all class data participating in AE training.

Algorithm 1 Proposed Autoencoder Threshold Calculation

$X \leftarrow$ Data of a specific class
$n_{samples} \leftarrow$ Number of X samples
$n_{features} \leftarrow$ Number of X features
$AE \leftarrow$ Trained autoencoder with X data
$th \leftarrow (0, 0, ..., 0_{n_{features}})$
for $i = 1$ to $n_{samples}$ **do**
  $\hat{X}_i \leftarrow AE(X_i)$
  $(r_1, r_2, ..., r_{n_{features}}) \leftarrow RE(X_i, \hat{X}_i)$
  $th \leftarrow max((th_1, th_2, ..., th_{n_{features}}), (r_1, r_2, ..., r_{n_{features}}))$
end for

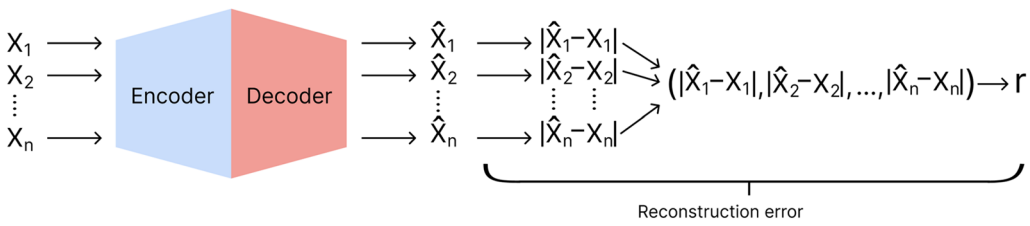

Fig. 5 The procedure of calculation RE in the proposed model

After selecting the threshold, which is a vector of the size of the features, we can perform the anomaly detection. We first enter any given data in the trained AE with normal data. The RE vector for the data is generated by calculating the RE of each feature. After calculating the RE, we compare it with the selected threshold value for that AE. If only one feature of the RE vector is greater than the corresponding value of that feature in the threshold vector, we consider the input data an anomaly. The pseudo-code of anomaly detection algorithm based on AE reconstruction error is shown in Algorithm 2.

Algorithm 2 Anomaly Detection Algorithm in Our Proposed Model

$X \leftarrow$ dataset
$n_{samples} \leftarrow$ Number of X samples
$n_{features} \leftarrow$ Number of X features
$AE \leftarrow$ trained autoencoder with specific class
$(th_1, th_2, ..., th_{n_{features}}) \leftarrow$ threshold of the autoencoder
for $i = 1$ to $n_{samples}$ **do**
  $\hat{X}_i \leftarrow AE(X_i)$
  $(r_1, r_2, ..., r_{n_{features}}) \leftarrow RE(X_i, \hat{X}_i)$
  $j \leftarrow 1, n_{features}$
  if all $r_j < th_j$ **then**
    X_i classify in the AE trained class
  else
    X_i does not classify in the AE trained class
  end if
end for

We also classify other classes in the dataset with the proposed method. First, AEs with different data from dataset classes are learned. Thresholds are selected according to the process mentioned in Algorithm 2 for each AE. AEs can then perform binary classification. Assuming input data, RE value is calculated, and according to RE and threshold value, the data can be classified as AE training data class or anomaly.

In terms of computational complexity, given the characteristics of the data, our proposed method takes more calculations to discover anomalies than current practises. This is because we calculate a threshold for each characteristic. We classify input data by comparing all features' reconstruction errors to the associated threshold. Of course, when comparing both algorithms in the same autoencoder network, this result is correct. Other factors, such as the number of hidden layers in the autoencoder network and the number of neurons in each layer, have a greater impact on the number of final calculations and network efficiency if the autoencoder networks. Therefore, the performance of the anomaly detection method in the autoencoder network is mostly influenced by the architecture of the autoencoder network.

Multi-class classification with hierarchical structure

Some binary classifiers may have poor performance due to several causes. For example, the amount of data in a given class may not be large enough for the training algorithm. Due to poor data set labeling, there may be classes in which data is inherently similar to other classes. This poor performance in a classifier may lead to many false-positive detections. One of the promising solutions to this issue is multi-class architecture. In the proposed multi-class architecture, we place the classifiers with better performance at the beginning, and the weaker ones are placed at the end of the architecture. The proposed architecture is shown in Fig. 6.

As illustrated in Fig. 6, the entire dataset is first presented to the classifier A. This classifier only classifies A data. Data classified as A are subtracted in the dataset and then presented to the classifier B. This process is repeated for the rest of the classifiers. The order of classifiers is determined by selecting the one with the best false-positive rate.

Evaluation results and discussion

Evaluation metrics

This section demonstrates the performance of the proposed model on the CIDDS-001 dataset. The codes used for the evaluation of the proposed model is publicly available through GitHub. The link is available in the "data availability statement" section. The following parameters are used to evaluate the performance of the proposed model:

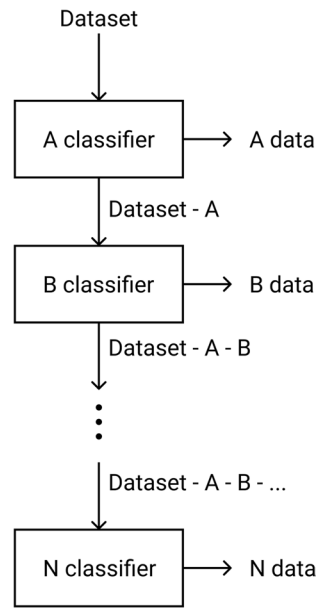

Fig. 6 Proposed multi-class classification architecture

- Detection Rate or Recall, defined in Eq. 7
- False Positive Rate (FPR), defined in Eq. 8
- Precision, defined in Eq. 9
- Accuracy, defined in Eq. 10
- F1-score, defined in Eq. 11

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

$$FPR = \frac{FP}{FP + TN} \tag{8}$$

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

$$F_1 Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{11}$$

In the Eqs. 7 to 11, the parameters *TP*, *TN*, *FP*, and *FN* refer to True Positive, True Negative, False Positive, and False Negative, respectively.

Availability of dataset

We use the CIDDS-001 [15] dataset to perform AE-based anomaly detection in cloud networks. CIDDS-001 is accepted as the benchmark dataset and contains unidirectional NetFlow data. It comprises data from two servers: OpenStack and an external server. The dataset is created by simulating a small business environment that includes an OpenStack with internal servers (web, file, backup, and mail) and an external server (file synchronization and web server) that is installed on the internet to capture real-time internet traffic [31]. The dataset contains traffic data from two servers, with each server's traffic consisting of four weeks of traffic data. Additionally, the original data of CIDDS-001 consists of 16 features, including Src IP, Src Port, Dest IP, Dest Port, Proto, Date first seen, Duration, Bytes, Packets, Flags, Tos, Flows, Class, AttackType, AttackID, and AttackDescription. The network attacks are categorized into five classes: normal, suspicious, unknown, attacker, and victim. In this paper, we use external traffic. We omit three features, including AttackID, AttackType, and AttackDescription, since they are more related to the attack's information than anomaly detection. This dataset contains many traffic instances, 153026 of which are used for the study from the External Server.

Pre-processing of data

We perform the following pre-processing steps on the dataset. AE networks only use numerical data for training and testing. Therefore, the first step is to convert nominal and categorical data into numerical data. All the nominal and categorical values are mapped into numeric values using one-hot encoding and binary encoding. After pre-processing of data, the dataset contains features of numerical values. The numerical values were normalized and arranged between 0 and 1 by using Eq. 12.

$$\bar{x_i} = \frac{x_i - min(x_i)}{max(x_i) - min(x_i)}, \quad \text{For} \quad i = 1, \ldots, n \tag{12}$$

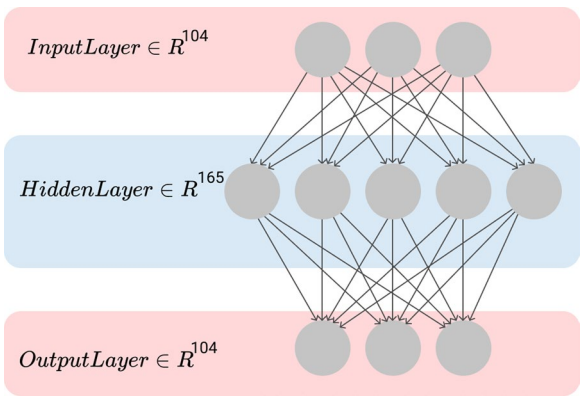

Fig. 7 The architectural diagram of AE model including the number of input, hidden and output layers

where n represents the number of records, and x represents a specific column in the dataset.

Performance evaluation

This section provides the configuration of AE, multi-class classification, and an analysis of the performance results of the proposed model. First, we perform a hyper-parameter optimization for AE networks with grid search, generating the best setting for hyper-parameters. The AE network is composed of one hidden layer with 165 units, as shown in Fig. 7, with the ReLU activation function, and the activation function of the output layer is Sigmoid. We use RMSprop optimizer with a learning rate set to 0.00014. We perform multiple comparison results with the methods presented in [19, 20].

We use a re-sampling mechanism to prevent overfitting in our validation method. We employ k-fold cross-validation, which is a re-sampling strategy in which the dataset is divided into k equal-sized sections. The n-subsets are drawn for testing in the n-th of the k-loopings, while the blend of the remaining parts represents the training set. In our evaluations we use 10-fold cross-validation.

After training AEs on all data classes of the dataset, we calculate the threshold vector for each AE. We further feed all dataset data to every AE for evaluation as shown in Fig. 8. The evaluation of each classifier AE is reported in Table 1 and illustrated in Fig. 9.

Table 1 Performance of the different classifiers in the proposed model in terms of F1-score, FRP, Recall, Precision, and Accuracy on five classes

F1-score	FPR	Recall	Precision	Accuracy	Class
100	0	100	100	100	Normal
99.941	0.008	100	99.882	99.993	Attacker
99.906	0.007	100	99.813	99.992	Victim
99.781	0.125	100	99.562	99.903	Unknown
80.540	85.699	100	67.42	69.1	Suspicious

The results presented in this section are derived from the implementation of the described models and metrics using the Python programming language. The used libraries are listed as follows:

- Numpy [32], and Pandas [33] are used for pre-processing and manipulation and for calculating evaluation metrics.
- Optuna [34] is used for hyper-parameter optimization.
- Pytorch [35] is used to implement AE.

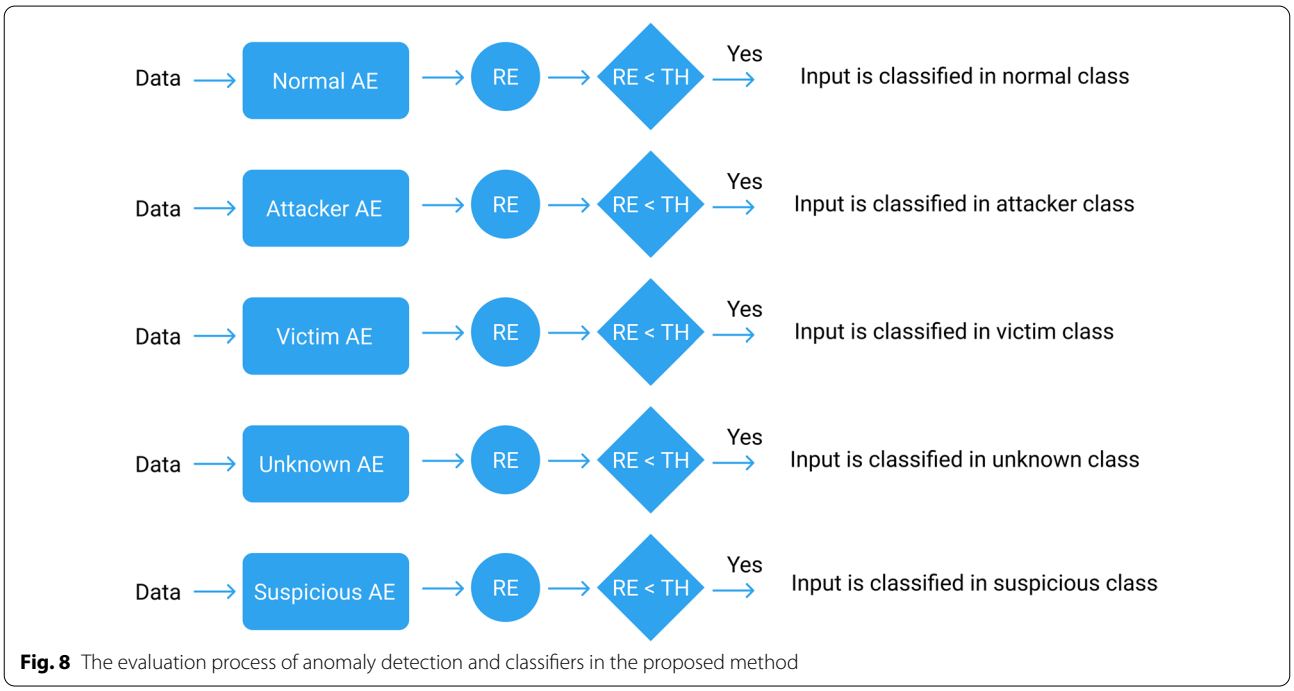

Fig. 8 The evaluation process of anomaly detection and classifiers in the proposed method

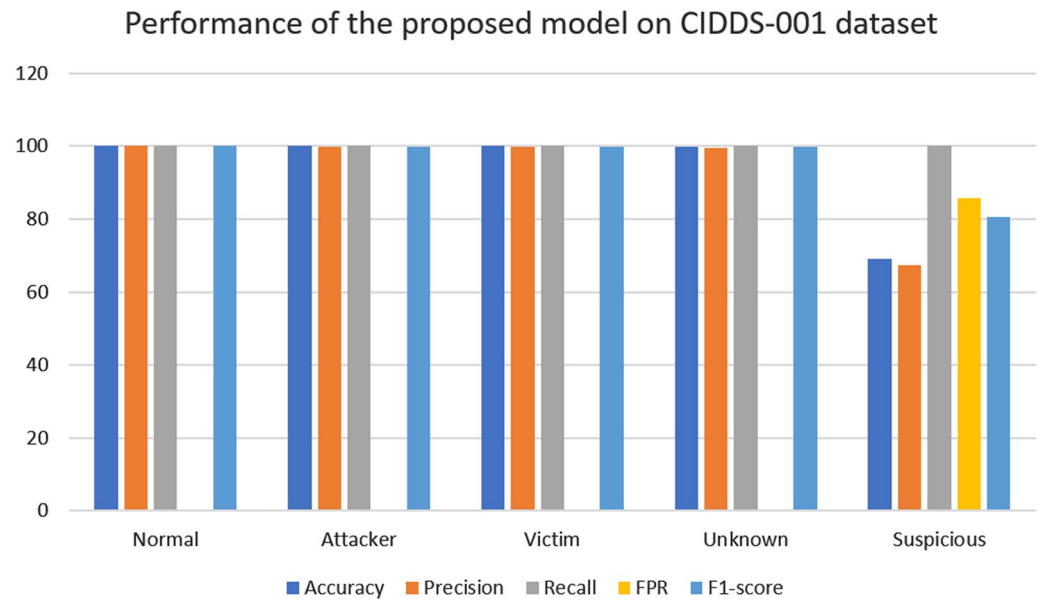

Fig. 9 Graphical representation for evaluation results of different classifiers in the proposed model on CIDDS-001 dataset for five classes of Normal, Attacker, Victim, Suspicious and Unknown for F1-score, FRP, Recall, Precision and Accuracy metrics

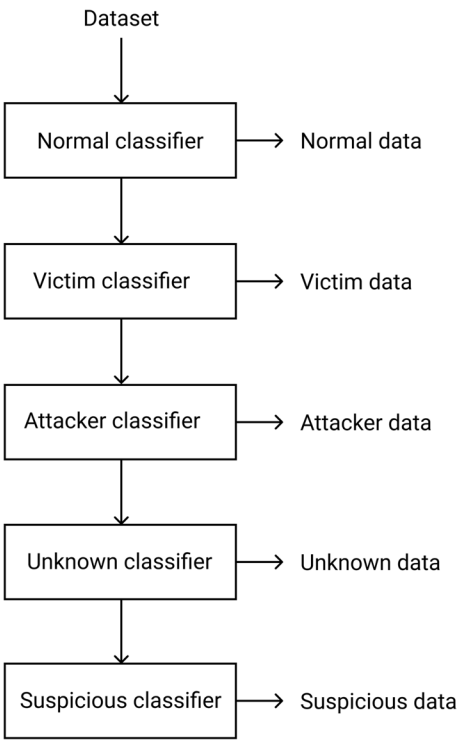

Fig. 10 The hierarchical structure for multi-class classification

The training procedure and evaluations are performed on Google Colab [36].

As can be seen in Table 1 and Fig. 9, the recall value of all classifiers is 1 or 100%. It happens because chosen

Table 2 Performance of the proposed model after using multi-class classification architecture in terms of F1-score, FRP, Recall, Precision, and Accuracy metrics

F1-score	FPR	Recall	Precision	Accuracy	Class
100	0	100	100	100	Normal
99.941	0.007	100	99.882	99.993	Attacker
99.906	0.007	100	99.813	99.992	Victim
99.792	0.118	100	99.585	99.907	Unknown
99.916	0	99.833	100	99.893	Suspicious

thresholds categorize the data in the correct class. The false-positive rate of all classifiers except the suspicious class is low. The reason for the high false-positive rate of the suspicious class is that there are issues in labeling the CIDDS-001 dataset. Network traffics received from the internet in dataset generation are labeled as suspicious [31]. Therefore, this class of data is not in control of dataset generation. We use the proposed multi-class classification architecture with a hierarchical structure to increase the classifiers' accuracy. The priority of the classifiers is with the classifiers with the lowest value of the false-positive rate. The classifier with the lowest false-positive rate is at the beginning of the architecture, and the classifier with the highest false-positive rate is at the end. Low false-positive rates are important because the classifier at the beginning of the architecture does not classify the data of other classes incorrectly. We now evaluate this architecture with all the data in the dataset

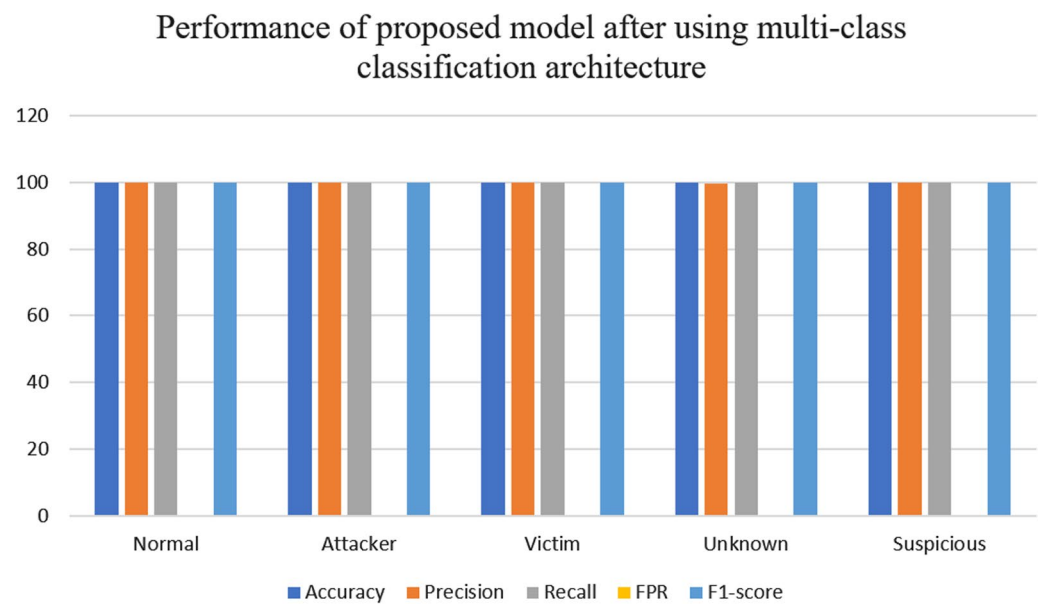

Fig. 11 Graphical representation for evaluation results of proposed model after using multi-class classification on CIDDS-001 dataset for five classes of Normal, Attacker, Victim, Suspicious and Unknown for F1-score, FRP, Recall, Precision and Accuracy metrics

as shown in Fig. 10. The evaluation results are reported in Table 2 and illustrated in Fig. 11.

As presented in Table 2, the proposed model achieves the good detection performance on anomaly detection or classifying normal data from other data in the CIDDS-001 dataset. It achieves exactly 100% accuracy, 100% precision, 100% recall, 0% FPR, and 100% F1-score.

Comparison results

Many research studies have been conducted in anomaly detection by using the CIDDS-001 dataset. A comparison between our proposed method and the methods presented [19] is illustrated in Table 3. In [19], the authors evaluate several state of the art algorithms. It means that comparing the results reported in [19] we are comparing the proposed method with 6 other methods. As results indicate, our method with simple AE consisting of one hidden layer has much better performance than the deep AE with multiple hidden layers presented in [19].

As shown in Table 3, our method is better in most evaluation metrics values. We also compare our proposed method, a simple AE, with the method presented in [20] and [27], which are Deep AE and stacked autoencoder based neural network in Tables 4 and 5. As results show, our proposed method performs best in anomaly detection or normal classification performed with the CIDDS-001 dataset.

The evaluation results of our proposed model based on AE are better than Deep AE. This shows the advantage of our approach. Based on these comparisons, the results obtained using the proposed model are very promising.

The architecture of autoencoder networks in [20] and [27] are all deep and have several hidden layers. Our presented autoencoder network in this paper has only one hidden layer and therefore, our proposed method is more efficient compared with [20] and [27].

Our proposed method uses the class data to calculate the classifier's threshold; therefore, if data from the trained class data is included in other classes, the data will be incorrectly classified. This issue could be the result of an error in the dataset's labelling or a flaw in the dataset's collection. As a result, when the same data exists in two separate classes, thresholding becomes difficult, and the capacity to detect anomalies or categorise through reconstruction mistakes is diminished. There could be researches to overcome this issue, however, we believe that one promising solution as a future direction to this research is to use variational autoencoder networks. Reconstruction probability can also be utilised for anomaly detection or classification in variational autoencoder networks.

0In the proposed solution, we choose a network design for all AEs. By supplying some data from all classes, the architecture is retrieved using hyper-parameter optimization. This limitation could be addressed in the future

Table 3 Performance comparison of the proposed model with methods presented in [19] including SVM, KNN, Naïve Bayes, NN, DNN and DAE

Recall	Precision	F1-score	Accuracy	Method	Class
0.509	0.186	0.272	0.515	SVM	Normal
0.997	0.973	0.985	0.994	KNN	
0.975	0.991	0.983	0.994	Naïve Bayes	
0	0	0	0.822	NN	
0	0	0	0.850	DNN	
0.805	0.810	0.795	0.822	DAE	
1	1	1	1	Our proposed model	
0.150	0.022	0.039	0.888	SVM	Attacker
0.997	0.979	0.988	1	KNN	
0.999	0.999	0.999	1	Naïve Bayes	
0	0	0	0.985	NN	
0	0	0	0.989	DNN	
0	0	0	0.985	DAE	
1	0.998	0.999	0.999	Our proposed model	
0.854	0.496	0.627	0.985	SVM	Victim
0.996	0.981	0.988	1	KNN	
0.999	0.999	0.999	0.999	Naïve Bayes	
0	0	0	0.985	NN	
0	0	0	0.985	DNN	
0	0	0	0.985	DAE	
1	0.998	0.999	0.999	Our proposed model	
0.307	0.064	0.106	0.689	SVM	Unknown
0.784	0.834	0.808	0.978	KNN	
0.880	0.603	0.715	0.958	Naïve Bayes	
0	0	0	0.940	NN	
0	0	0	0.967	DNN	
0	0	0	0.940	DAE	
1	0.995	0.997	0.999	Our proposed model	
1	0.754	0.177	0.318	SVM	Suspicious
0.980	0.982	0.981	0.972	KNN	
0.952	0.985	0.968	0.954	Naïve Bayes	
1	0.732	0.845	0.732	NN	
1	0.752	0.885	0.710	DNN	
1	0.732	0.845	0.732	DAE	
0.998	1	0.999	0.998	Our proposed model	

Table 5 Performance comparison of the proposed model with the method presented in [27] in terms of Precision and F1-score

Class	Method	F1-score	Precision
Normal	Stacked autoencoder based neural network	98.93	99.05
	Our proposed method	100	100

researches by determining the ideal architecture of autoencoder network of each class in hyper-parameter optimization using only that class's data.

Conclusions

This paper investigates anomaly detection in cloud network data by using autoencoders. A new approach for autoencoder-based anomaly detection was proposed. This approach is different from previous autoencoder-based anomaly detection techniques in terms of the reconstruction error calculation mechanism, threshold determination, and the RE compared with the threshold method. The parameters of our proposed model were tuned using the grid search technique. We also proposed a multi-class classification architecture with a hierarchical structure to enhance the performance of the proposed model. We made evaluations by using different metrics, including accuracy, precision, recall, false-positive rate, and F1-score on a commonly used dataset named CIDDS-001. By analyzing the evaluation results and comparing them with some of the recent existing methods, it was shown that the proposed anomaly detection had a better performance than the existing ones. The results confirmed that the proposed approach could be a promising solution, with high efficiency for constructing autoencoder-based anomaly detection for cloud networks. The codes used for the evaluation are publicly available and could be used by researchers for further investigations. Although the proposed method performs well in simulations and benchmark datasets, issues still appear in practical cases. This research could be resumed by applying the proposed algorithm in practical evaluations and real networks.

Table 4 Performance comparison of the proposed model with the method presented in [20] in terms of F1-score, Recall, Precision, FPR, and Accuracy

Class	F1-score	Recall	Precision	Accuracy	FPR	Method
Normal	95	100	90	97	4	DAE
	100	100	100	100	0	Our proposed method

Abbreviations
DNN: Deep neural network; IT: Information technology; IoT: Internet of things; LSTM: Long-short term memory; SVM: Support vector machines; NB: Naïve Bayes; MLP: Multi-layer perceptron; KNN: k-nearest neighbors; NN: Neural networks; DAE: Denoising autoencoder; GWO: Grey wolf optimization; CNN: Convolutional neural network; SDN-ADS: Software-defined network-based anomaly detection system; API: Programming interface; ROC: Region of convergence; AUC: Area under the curve; AE: Autoencoder; RE: Reconstruction error.

Acknowledgements
Not applicable.

Author contributions
All authors read and approved the final manuscript.

Funding
Not applicable. This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Availability of data and materials
Dataset used in paper is publicly available in [15]. The codes used for evaluations are also publicly available at https://Github.Com/Hasan-Torabi/Anomaly-Detection-in-Cloud-Computing-Networks-by-Using-Autoencoders/.

Declarations

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Department of Electrical and Computer Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran. [2]Department of Informatics, Modeling, Electronics and System Engineering, University of Calabria, Arcavacata, Italy.

Received: 10 August 2022 Accepted: 12 December 2022
Published online: 04 January 2023

References

1. Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I et al (2010) A view of cloud computing. Commun ACM 53(4):50–58
2. Sagar S, Keke C (2021) Confidential machine learning on untrusted platforms: a survey. Cybersecurity 4(1):1–19
3. Sadiku MN, Musa SM, Momoh OD (2014) Cloud computing: opportunities and challenges. IEEE Potent 33(1):34–36
4. Popa L, Kumar G, Chowdhury M, Krishnamurthy A, Ratnasamy S, Stoica I (2012) Faircloud: sharing the network in cloud computing. In: Proceedings of the ACM SIGCOMM 2012 conference on applications, technologies, architectures, and protocols for computer communication, pp 187–198
5. Gupta R (2012) Above the clouds: a view of cloud computing
6. Alam T (2020) Cloud computing and its role in the information technology. IAIC Trans Sustain Digital Innov (ITSDI) 1(2):108–115
7. Hussein NH, Khalid A (2016) A survey of cloud computing security challenges and solutions. Int J Comput Sci Inf Secur 14(1):52
8. Hong JB, Nhlabatsi A, Kim DS, Hussein A, Fetais N, Khan KM (2019) Systematic identification of threats in the cloud: a survey. Comput Netw 150:46–69
9. Ometov A, Molua OL, Komarov M, Nurmi J (2022) A survey of security in cloud, edge, and fog computing. Sensors 22(3):927
10. Alijani GS, Fulk HK, Omar A, Tulsi R (2014) Cloud computing effects on small business. Entrep Execut 19:35
11. Soldani J, Brogi A (2022) Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: a survey. ACM Comput Surv (CSUR) 55(3):1–39
12. Yasarathna TL, Munasinghe L (2020) Anomaly detection in cloud network data. In: 2020 International research conference on smart computing and systems engineering (SCSE). IEEE, pp 62–67
13. Disha RA, Waheed S (2022) Performance analysis of machine learning models for intrusion detection system using gini impurity-based weighted random forest (giwrf) feature selection technique. Cybersecurity 5(1):1–22
14. Islam MS, Miranskyy A (2020) Anomaly detection in cloud components. In: 2020 IEEE 13th international conference on cloud computing (CLOUD). IEEE, pp 1–3
15. Ring M, Wunderlich S, Grüdl D, Landes D, Hotho A (2017) Flow-based benchmark data sets for intrusion detection. In: Proceedings of the 16th European conference on cyber warfare and security. ACPI, pp 361–369
16. Qureshi KN, Jeon G, Piccialli F (2021) Anomaly detection and trust authority in artificial intelligence and cloud computing. Comput Netw 184:107647
17. Tama BA, Rhee K-H (2017) Attack classification analysis of iot network via deep learning approach. Res Briefs Inf Commun Technol Evol (ReBICTE) 3:1–9
18. Althubiti SA, Jones EM, Roy K (2018) Lstm for anomaly-based network intrusion detection. In: 2018 28th international telecommunication networks and applications conference (ITNAC). IEEE, pp 1–3
19. Rashid A, Siddique MJ, Ahmed SM (2020) Machine and deep learning based comparative analysis using hybrid approaches for intrusion detection system. In: 2020 3rd international conference on advancements in computational sciences (ICACS). IEEE, pp 1–9
20. Fenanir S, Semchedine F, Harous S, Baadache A (2020) A semi-supervised deep auto-encoder based intrusion detection for iot. Ingénierie des Systèmes d'Information 25(5)
21. Garg S, Kaur K, Kumar N, Kaddoum G, Zomaya AY, Ranjan R (2019) A hybrid deep learning-based model for anomaly detection in cloud datacenter networks. IEEE Trans Netw Serv Manag 16(3):924–935
22. Ge M, Syed NF, Fu X, Baig Z, Robles-Kelly A (2021) Towards a deep learning-driven intrusion detection approach for internet of things. Comput Netw 186:107784
23. Verma A, Ranga V (2018) Statistical analysis of cidds-001 dataset for network intrusion detection systems using distance-based machine learning. Procedia Comput Sci 125:709–716
24. Albahar MA, Binsawad M, Almalki J, El-etriby S, Karali S (2020) Improving intrusion detection system using artificial neural network. Int J Adv Comput Sci Appl 11(6)
25. Zavrak S, Skefiyeli M (2020) Anomaly-based intrusion detection from network flow features using variational autoencoder. IEEE Access 8:108346–108358
26. Oliveira N, Praça I, Maia E, Sousa O (2021) Intelligent cyber attack detection and classification for network-based intrusion detection systems. Appl Sci 11(4):1674
27. Tao X, Kong K, Zhao F, Cheng S, Wang S (2020) An efficient method for network security situation assessment. Int J Distrib Sens Netw 16(11):1550147720971517
28. Hinton GE, Zemel R (1993) Autoencoders, minimum description length and helmholtz free energy. Adv Neural Inf Process Syst 6
29. Bourlard H, Kamp Y (1988) Auto-association by multilayer perceptrons and singular value decomposition. Biol Cybern 59(4):291–294
30. Sakurada M, Yairi T (2014) Anomaly detection using autoencoders with nonlinear dimensionality reduction. In: Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis, pp 4–11
31. Ring M, Wunderlich S, Gruedl D, Landes D, Hotho A (2017) Technical report cidds-001 data set. J Inf Warfare 13
32. Harris CR, Millman KJ, Van Der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ et al (2020) Array programming with numpy. Nature 585(7825):357–362
33. McKinney W (2015) Pandas, python data analysis library. http://pandas.pydata.org
34. Akiba T, Sano S, Yanase T, Ohta T, Koyama M (2019) Optuna: a next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pp 2623–2631
35. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L et al (2019) Pytorch: an imperative style, high-performance deep learning library. Adv Neural Inf Process Syst 32
36. Bisong E (2019) Building machine learning and deep learning models on google cloud platform. Springer

Publisher's Note