

## Article

# A Multi-Channel Contrastive Learning Network Based Intrusion Detection Method

Jian Luo <sup>1,\*</sup> , Yiying Zhang <sup>1</sup>, Yannian Wu <sup>2</sup>, Yao Xu <sup>1</sup>, Xiaoyan Guo <sup>3</sup> and Boxiang Shang <sup>4</sup>

<sup>1</sup> Department of Internet of Things Engineering, Tianjin University of Science and Technology, Tianjin 300457, China

<sup>2</sup> Shenzhen Guodian Technology Communication Co., Shenzhen 518028, China

<sup>3</sup> Information and Communication Company, State Grid Tianjin Electric Power Company, Tianjin 300140, China

<sup>4</sup> State Grid Tianjin Electric Power Company, Tianjin 300131, China

\* Correspondence: steelknife@163.com

**Abstract:** Network intrusion data are characterized by high feature dimensionality, extreme category imbalance, and complex nonlinear relationships between features and categories. The actual detection accuracy of existing supervised intrusion-detection models performs poorly. To address this problem, this paper proposes a multi-channel contrastive learning network-based intrusion-detection method (MCLDM), which combines feature learning in the multi-channel supervised contrastive learning stage and feature extraction in the multi-channel unsupervised contrastive learning stage to train an effective intrusion-detection model. The objective is to research whether feature enrichment and the use of contrastive learning for specific classes of network intrusion data can improve the accuracy of the model. The model is based on an autoencoder to achieve feature reconstruction with supervised contrastive learning and for implementing multi-channel data reconstruction. In the next stage of unsupervised contrastive learning, the extraction of features is implemented using triplet convolutional neural networks (TCNN) to achieve the classification of intrusion data. Through experimental analysis, the multichannel contrastive learning network-based intrusion-detection method achieves 98.43% accuracy in dataset CICIDS17 and 93.94% accuracy in dataset KDDCUP99.

**Keywords:** network intrusion detection; feature reconstruction; autoencoder; multi-channel; contrastive learning



**Citation:** Luo, J.; Zhang, Y.; Wu, Y.; Xu, Y.; Guo, X.; Shang, B. A Multi-Channel Contrastive Learning Network Based Intrusion Detection Method. *Electronics* **2023**, *12*, 949. <https://doi.org/10.3390/electronics12040949>

Academic Editors: Phivos Mylonas, Katia Lida Kermanidis and Manolis Maragoudakis

Received: 11 January 2023

Revised: 4 February 2023

Accepted: 6 February 2023

Published: 14 February 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The network is a national infrastructure and one of the primary targets of attack in modern warfare, where defense against cyber attacks has become a growing concern for researchers. In 2020, Brazil's Light S.A. electricity company was hacked to extort \$14 million in ransom, and in late February 2022, the Internet was frequently attacked and controlled from abroad, and cross-space cyber attacks were carried out against Russia and Ukraine. A network intrusion-detection system is achieved by analyzing the characteristics of network data streams to determine the network streams as normal data streams and attack data streams. Intrusion-detection systems are still challenging in the face of the high dimensionality of data features and extreme imbalance of intrusion categories, and the systems exhibit low accuracy and high false alarm rates. To solve the above problems, numerous researchers have mainly focused on machine learning methods [1], deep learning methods [2], and contrastive learning [3].

Various machine learning and deep learning-based solutions have been proposed in the past decades. Among them, machine learning-based network intrusion network detection systems rely mainly on feature engineering, so as to learn information about the characteristics of network intrusion data [4]. Deep learning-based network intrusion-detection approaches, on the other hand, do not rely on huge feature engineering, but

instead learn the complex features of network intrusion data from deep network structures [5].

As the application of supervised deep learning continues to evolve, deep learning has shown a significant decrease in performance when dealing with data imbalance [6], and unsupervised contrastive learning continues to narrow the gap between supervised deep learning. The purpose of contrastive learning is to achieve judgments on predicted samples by reducing the distance between like classes and increasing the distance between different classes and by computing the distance (e.g., Euclidean distance). Contrastive learning is a combination of deep learning hierarchical learning features and self-defined sample distances to deal with data imbalance.

Autoencoder is widely used to attract our interest. Autoencoder as a kind of neural network, whose architecture includes an input layer, encoder, hidden layer, decoder, and output layer, implements recoding the raw data according to the label of the data and minimizing a loss function. Recently, autoencoder is widely used in data dimensionality reduction, data reconstruction, and data noise reduction. For example, in [7], an autoencoder is used to achieve feature extraction from the raw data and shows good experimental performance. As in the literature [8], the decoder of the autoencoder is used for noise reduction in the raw data. Moreover, the autoencoder is simple to train and continues to be more efficient.

In contrastive learning, many of the state-of-the-art deep neural networks are used in contrastive learning [9,10]. In the literature [11], it is proposed that the detection effectiveness of a network intrusion-detection model depends on the loss function of contrastive learning and is one of the important components of the model. In network intrusion detection, intrusion data and anomaly data represent only a small fraction of the network data [12]. In the literature [13], the researchers studied the resampling of deep neural networks, thus verifying that neural network algorithms are robust in dealing with data imbalance.

Multichannel feature extraction is applied in the tasks of image analysis [14] and speech recognition [15] to improve the accuracy of the model by learning to correlate between different channels. During multi-channel data enhancement, our method uses the network data stream as the raw data vector and dichotomizes the data according to the labels of the raw data, which are normal and attack network streams. The normal autoencoder is trained using normal data, and the attack autoencoder is trained using attack data. The raw data are fed into the corresponding autoencoder according to the labels to obtain a one-dimensional embedding representation of the output. We compute the cross-correlation matrix for the one-dimensional data of the autoencoder, and reconstruct the different cross-correlation matrices to obtain the multi-channel data and use it as a new description of the raw data. The multi-channel data can represent the connection existing between different features and enrich the features of the raw data. In this paper, the extraction of the features of the one-dimensional embedding representation is transformed into the extraction of the features of the multi-channel cross-correlation matrix, which increases the gap between different classes of data and improves the accuracy of model detection.

Contrastive learning applications have recently received attention in image classification by setting innovative loss functions to improve the poor learning performance of the model in addressing data imbalance. However, these methods are usually applied in the feature extraction of multi-channel images. Therefore, we use TCNN as a contrastive learning network for feature-reconstruction multichannel two-dimensional vector-data feature extraction to reduce the distance between like classes and increase the distance between different classes to improve the accuracy of network intrusion detection.

The main innovations of the proposed model are as follows.

- (a) A new network intrusion-detection model is proposed to recode the network intrusion data using autoencoder according to the labels, realizing autoencoder coupled with TCNN, which has high accuracy and low false alarm rate and improves the security of intrusion detection.

- (b) The features are augmented, and the raw single-channel one-dimensional data are feature-enhanced into multi-channel two-dimensional data.
- (c) The problem of extreme data imbalance encountered in network intrusion is solved, and we evaluate the model extensively.

## 2. Related Research

Network intrusion detection is often viewed as a binary classification problem, where the classification of network data streams is achieved by setting the model's feature extraction method and the model's classification rules, among which machine learning methods such as k-Nearest Neighbor (KNN), support vector machine (SVM), decision tree (DT), etc., are used. Zhou et al. proposed an intrusion-detection method that is based on the selection of the most relevant features, and an integrated classifier based on Random Forest (RF), C4.5, and Penalized Attribute Forest (Forest PA), and finally, classification is achieved by voting technique [16]. While traditional machine learning algorithms enable network intrusion detection, these methods have low accuracy rates during experiments.

Deep learning methods learn to extract data features by a hierarchical approach, which enables the extraction of high-dimensional features from raw data. Deep learning is currently used with remarkable effects in image recognition [17] and sentiment analysis [15]. Autoencoders are widely used in deep learning, and Zeng et al. stacked autoencoders and used the output of the previous layer of autoencoding as the input of the next layer [18]. Sara A. Althubiti et al. applied the Long Short-Term Memory algorithm (LSTM) to a network intrusion-detection system, and validated their model on the CICIDS dataset, with results demonstrating deep learning algorithms outperform machine learning methods [19]. Lopez et al. [20] used a one-dimensional convolutional neural network to achieve better experimental results for feature extraction of one-dimensional network intrusion data. Deep learning-based network intrusion-detection algorithms are able to achieve high accuracy rates, but are ineffective in handling network intrusion data imbalance experiments.

The contrastive learning approach uses a hierarchical learning approach to achieve a transformation of the raw data to map the raw data to a suitable feature space. Contrastive learning is implemented by deep neural networks, as well as by defining a sample distance loss function to learn different classes of sample features in order to alleviate the prediction error under data imbalance. Currently, contrastive learning is mainly used in face recognition and face verification in [21]. As pointed out in the literature [22], the performance of contrastive learning networks depends mainly on the defined loss function (e.g., contrastive loss, triplet loss) and the network sampling method. In the above contrastive learning method, no data preprocessing is performed, and the training process results in a model that is not optimal.

Novel feature extraction models have been reported in recent cybersecurity research. In the literature [23], an autoencoder was used to learn the raw data features, and a deep neural network was used to extend the new feature data to compose multi-channel data from the raw data and the new feature data to train a multi-channel convolutional neural network. In the literature [24], based on the combination of autoencoder and contrastive learning, it is demonstrated that contrastive learning has good performance in dealing with data imbalance and nonlinear data structure problems. Therefore, we propose an auto-encoder to pre-process the data and reconstruct the multi-channel data to increase the differences between different categories and finally implement a contrast learning network intrusion-detection method.

## 3. Model Methodology

In this section, we describe MCLDM—our proposed multichannel contrastive learning method for implementing network intrusion detection—which is a combination of a supervised contrastive learning method (two autoencoders) and an unsupervised contrastive learning method for multichannel feature extraction (TCNN). The symbols used in the MCLDM are shown in the following Table 1.

**Table 1.** Symbol representation.

Symbol	Description
$X$	training data matrix $X \subset \mathbb{R}^D$
$X^n$	The labels in $X$ are a subset of the normal samples
$X^a$	The labels in $X$ are a subset of the attack samples
$g_n$	normal autoencoder trained on $X^n$
$g_a$	attack autoencoder trained on $X^a$
$X^{n+}$ $X^{n-}$	$X^n$ Output via autoencoder $g_n$ $g_a$
$X^{a+}$ $X^{a-}$	$X^a$ Output via autoencoder $g_n$ $g_a$
$X_{cor}^n$ $X_{cor}^a$	the crosscorrelation matrix with $X^n$
$X_{cor}^{n+}$ $X_{cor}^{a+}$	the crosscorrelation matrix with $X^{n+}$
$X_{cor}^{n-}$ $X_{cor}^{a-}$	the crosscorrelation matrix with $X^{n-}$
$[x_{cor}^n, x_{cor}^n]$ $[x_{cor}^a, x_{cor}^a]$	representative anchor samples
$[x_{cor}^n, X_{cor}^{n+}]$ $[x_{cor}^a, X_{cor}^{a+}]$	representative positive samples
$[x_{cor}^n, X_{cor}^{n-}]$ $[x_{cor}^a, X_{cor}^{a-}]$	representative negative samples
$\Phi : \mathbb{R}^{D \times D} \rightarrow \mathbb{R}^d$	embedding space learned via a TCNN

Two stages are included in this MCLDM: the training stage and the prediction stage. In the training stage, the purpose of the MCLDM is mainly to learn the features of the raw vector of network intrusion data, train two types of autoencoders, realize the recoding of the normal network stream and the attack network stream, and output the one-dimensional reconstructed vector. The reconstructed vector and the raw vector are computed separately for the cross-correlation matrix, and the cross-correlation matrix is combined to form the multi-channel reconstructed data. Finally, the multi-channel reconstructed data are used as input to train the TCNN to achieve unsupervised contrastive learning according to the objective function, and finally the embedding of the comparison output and the calculation of the loss function to achieve the training of the MCLDM. In the prediction stage, the predicted data are input to the MCLDM to obtain the final embedding, and the type of data stream is determined by analyzing the embedding.

As shown in Figure 1, in the MCLDM training stage, (1) training data set  $X$  is divided into normal sample set  $X^n$  and attack sample set  $X^a$  by labeling the data. (2) Normal sample  $X^n$  and attack sample  $X^a$  are used as inputs to train autoencoder  $g_n$  and  $g_a$ , respectively. (3) Normal sample  $X^n$  and attack sample  $X^a$  are input to  $g_n$  and  $g_a$  to obtain  $[X^{n+}, X^{n-}]$  and  $[X^{a+}, X^{a-}]$ , respectively. (4) Triples  $[X^n, X^{n+}, X^{n-}]$  and  $[X^a, X^{a+}, X^{a-}]$  are constructed. (5) Triples  $[X^n, X^{n+}, X^{n-}]$  and  $[X^a, X^{a+}, X^{a-}]$  are obtained by combining them to obtain the cross-correlation matrix triplet  $[x_{cor}^n, X_{cor}^{n+}, X_{cor}^{n-}]$  and  $[x_{cor}^a, X_{cor}^{a+}, X_{cor}^{a-}]$ , which is combined to obtain the multichannel triplet  $([x_{cor}^n, x_{cor}^n], [x_{cor}^n, X_{cor}^{n+}], [x_{cor}^n, X_{cor}^{n-}])$ ,  $([x_{cor}^a, x_{cor}^a], [x_{cor}^a, X_{cor}^{a+}], [x_{cor}^a, X_{cor}^{a-}])$ . (6) The multichannel triplet is used as the input of the TCNN to learn the vector features of the training set. (7) The triplet loss is obtained by calculating different class-embedding representations to realize the training of the model in the state of data type imbalance.

### 3.1. Training Stage

The training stage of MCLDM pseudocode is described in Algorithm 1. This stage is to analyze the historical network intrusion data, learn the network intrusion data features, enrich the data features and reconstruct the multi-channel data, map the raw data to different vector spaces, and to realize to distinguish the normal network flow from the attack network flow. Specifically, three main stages are included in MCLDM.

- (1) By constructing two independent autoencoders and training the autoencoders according to the binary labels, where the labels are normal data streams and attack data streams, we achieve mapping the different autoencoder output data vectors into a vector space different from the raw data distribution.
- (2) Different autoencoders output reconstruction vectors, calculate reconstruction vector cross-correlation matrix, and combine different cross-correlation matrix arrays to

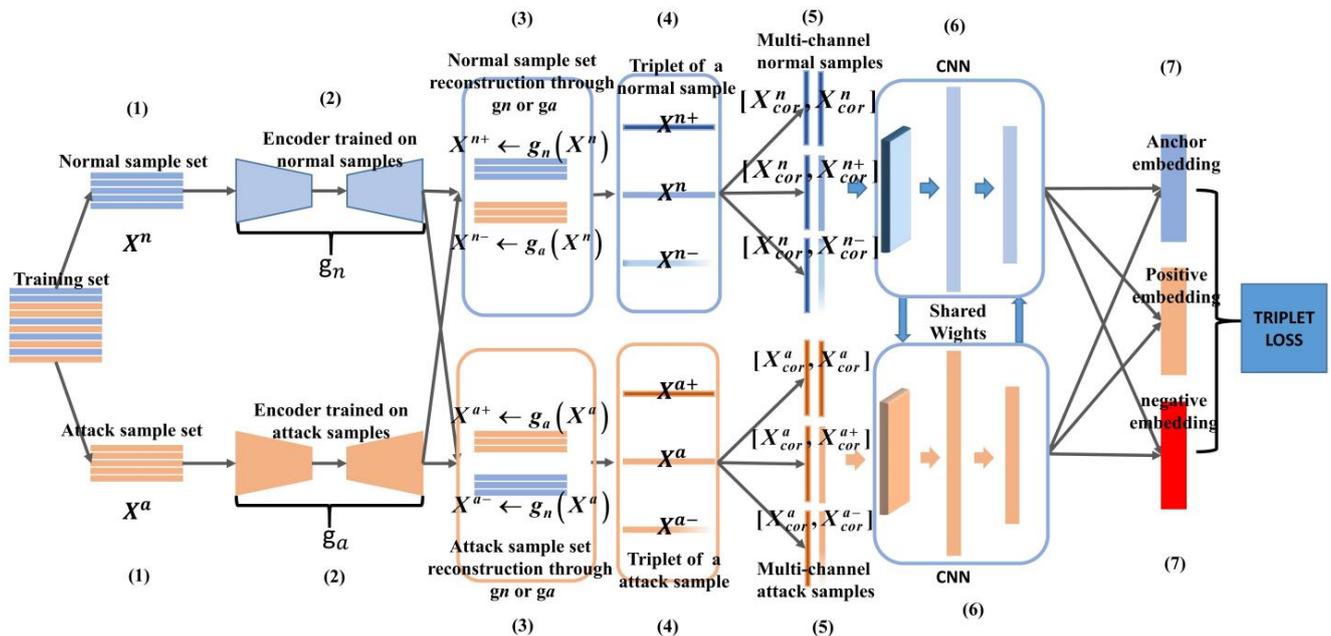
- obtain different multi-channel vector data, which include anchor points, positive samples and negative samples. The multi-channel vector data are formed into a triplet.
- (3) Train TCNN using the reconstructed ternary vectors in the previous stage.

**Algorithm 1** MCLDM training stage

**Input:**  $D$ : training sample set  $\{(X_i, label_i)\}_{i=1}^N$  with  $label_i \in \{normal, attack\}$ .  $X$  represents the raw data matrix  $N$  D-dimensional variables  $X \subset \mathbb{R}^D$ ,  $X_n \subset X_{label=normal}$  and  $X_a \subset X_{label=attack}$ .

**Output:**  $(g_n, g_a, \phi)$ : the trained intrusion-detection model

- 1 **Begin:** Initialize parameters
- 2 **#Autoencoder training stage**
- 3  $g_n \leftarrow train\ Autoencoder(X_n)$
- 3  $g_a \leftarrow train\ Autoencoder(X_a)$
- 4 **#Multi-channel data construction**
- 5 **Foreach**  $(X_n, label) \in D$  **do**
- 6     **If**  $y = normal$  **then**
- 7         1 |  $X^{n+} \leftarrow g_n(X^{n+}), X^{n-} \leftarrow g_n(X^{n-})$
- 8         **else**
- 9             2 |  $X^{a+} \leftarrow g_a(X^{a+}), X^{a-} \leftarrow g_a(X^{a-})$
- 10              $[X_{cor}^n, X_{cor}^{n+}, X_{cor}^{n-}] \leftarrow [X^n, X^{n+}, X^{n-}]$
- 11              $[X_{cor}^a, X_{cor}^{a+}, X_{cor}^{a-}] \leftarrow [X^a, X^{a+}, X^{a-}]$
- 12             Triplet  $\leftarrow [ \begin{matrix} ([x_{cor}^n, x_{cor}^n], [x_{cor}^n, X_{cor}^{n+}], [x_{cor}^n, X_{cor}^{n-}]) \cup \\ ([x_{cor}^a, x_{cor}^a], [x_{cor}^a, X_{cor}^{a+}], [x_{cor}^a, X_{cor}^{a-}]) \end{matrix} ]$
- 13     **#TCNN training stage**
- 14      $\Phi \leftarrow train\ TCNN(Triplet)$
- 15 **Return**  $g_n, g_a, \Phi$



**Figure 1.** Model training stage.

3.1.1. Autoencoder Training Stage

Autoencoders are deep neural networks with an architecture that learns the features of the data through encoders and decoders [17]. Autoencoder mainly consists of an input layer, encoder, hidden variable, decoder, and output layer. The data are fed to the encoder through the input layer and the encoder is converted to the hidden variable dimension by compressing the encoding of the raw vector data. The decoder performs decoding operations on the compressed data and outputs the reconstructed vector at the output layer. The autoencoder principle mainly consists of two stages: encoder  $f$ —input vector  $X$

by mapping into the hidden variable  $\tilde{X}$ , denoted as  $\tilde{X} = f(X)$ ; and decoder  $f'$ —hidden variable  $Y$  by mapping into the output space  $\hat{X}_n$ , denoted as  $\hat{X}_n = f'(\tilde{X})$ .

We select a set of  $N$  training data samples from the training set denoted as  $\mathfrak{R}^d = \{(X_i, label_i)\}_i^N$ , where  $X_i \subset \mathfrak{R}^D$  denotes the one-dimensional vector representation of the corresponding training data  $i$  samples  $D$  features, and  $label_i$  denotes the label information of the corresponding samples, and where  $X^n = \{X_i, label = normal\}$  and  $X^a = \{X_i, label = attack\}$  denote the normal network flow and the attack network flow, respectively. We train two independent autoencoders using  $X^n$  and  $X^a$ , respectively, where the normal network flow encoder is denoted as  $g_n$  and the attack network flow encoder is denoted as  $g_a$ . In real scenarios where the attack samples are much fewer than normal samples, we train different autoencoders to cope with a data imbalance.

### 3.1.2. Multi-Channel Data Construction

The raw vector data is reconstructed by the autoencoder and is denoted as  $\hat{X}_n$ . In principle, the reconstructed vector  $\hat{X}_n$  is more concentrated and less noisy than the raw vector  $X$ . We input the raw vector  $X$  to the autoencoders  $g_n$  and  $g_a$  to obtain  $X^+$ , and  $X^-$ . By computing the cross-correlation matrix of  $X$ ,  $X^+$ , and  $X^-$ , denoted as  $X_{cor}$ ,  $X_{cor}^+$ , and  $X_{cor}^-$ , we combine the three cross-correlation matrices to obtain the multi-channel data  $([X_{cor}, X_{cor}], [X_{cor}, X_{cor}^+], [X_{cor}, X_{cor}^-])$ , and build a triplet training sample. We denote  $[X_{cor}, X_{cor}]$  as an Anchor sample,  $[X_{cor}, X_{cor}^+]$  as a Positive sample, and  $[X_{cor}, X_{cor}^-]$  as a Negative sample. When  $label = normal$ , we specify  $[X_{cor}^n, X_{cor}^n]$  as the Anchor sample,  $[X_{cor}^n, X_{cor}^{n+}]$  as the Positive sample, and  $[X_{cor}^n, X_{cor}^{n-}]$  as the Negative sample. When  $label = abnormal$ , we specify  $[X_{cor}^a, X_{cor}^{a+}]$  as the Anchor sample,  $[X_{cor}^a, X_{cor}^{a-}]$  as the Positive sample, and  $[X_{cor}^a, X_{cor}^{a+}]$  as the Negative sample. By multi-channel data reconstruction, the feature extraction of one-dimensional vectors is transformed into the extraction of two-dimensional multi-channel data, increasing the gap between different categories and helping to cope with training data imbalance.

### 3.1.3. TCNN Training Stage

The convolutional neural network consists of an input layer, a convolutional layer, a pooling layer, a fully-connected layer, and an output layer. The convolutional layer extracts the feature information from the input layer, the pooling layer aims to reduce the dimensionality of the input data, and the fully connected layer aims to flatten the two-dimensional data into a one-dimensional vector. We use the constructed triplet  $([X_{cor}, X_{cor}], [X_{cor}, X_{cor}^+], [X_{cor}, X_{cor}^-])$  to train a TCNN. In this paper, where we are dealing with multichannel reconstruction data and the convolutional neural network, we choose AlexNet, where the TCNN includes AlexNet and fully connected layers. The TCNN processes  $([X_{cor}, X_{cor}], [X_{cor}, X_{cor}^+], [X_{cor}, X_{cor}^-])$  and is a shared network weight feedforward network. In this stage, we use the triplet loss function proposed in [10] to train MCLDM, as shown in Equation (1).

$$\begin{aligned} Dis_{ap} &= \|\varphi([X_{cor}, X_{cor}]) - \varphi([X_{cor}, X_{cor}^+])\|^2 \\ Dis_{an} &= \|\varphi([X_{cor}, X_{cor}]) - \varphi([X_{cor}, X_{cor}^-])\|^2 \\ Loss &= \log(1 + \exp(Dis_{ap} - Dis_{an})) \end{aligned} \quad (1)$$

The TCNN minimizes the distance between the anchor samples and the positive samples, i.e., minimizes  $Dis_{ap}$  and maximizes the distance between the anchor and the negative samples, i.e., maximizes  $Dis_{an}$ . In traditional neural network algorithms, classification is achieved by predicting the probability of a category, but the predicted probability does not work well when dealing with data imbalance. Therefore, converting the probability of predicted categories into predicted distances mitigates the effect of data imbalance on the model, where  $Dis_{an}$  and  $Dis_{ap}$  are denoted as the final output Euclidean distances.

### 3.2. Prediction Stage

In the prediction stage as in Figure 2, the test sample  $Y^a$  of the query is input to the autoencoder  $g_n$  and  $g_a$  output  $[Y^n, Y^{n+}, Y^{n-}]$ , the corresponding cross-correlation matrix  $[Y_{cor}^a, Y_{cor}^{a+}, Y_{cor}^{a-}]$  is calculated, and the multi-channel triplet  $([Y_{cor}^a, Y_{cor}^a], [Y_{cor}^a, Y_{cor}^{a+}], [Y_{cor}^a, Y_{cor}^{a-}])$  is obtained by combination, and the multi-channel triplet is input to the TCNN to calculate the distance of the output triplet-embedding representation, and finally, the prediction category is judged according to the distance.

$$Dis_n = \|\varphi([X_{cor}, X_{cor}]) - \varphi([X_{cor}, X_{cor}^+])\|^2 \tag{2}$$

$$Dis_a = \|\varphi([X_{cor}, X_{cor}]) - \varphi([X_{cor}, X_{cor}^-])\|^2 \tag{3}$$

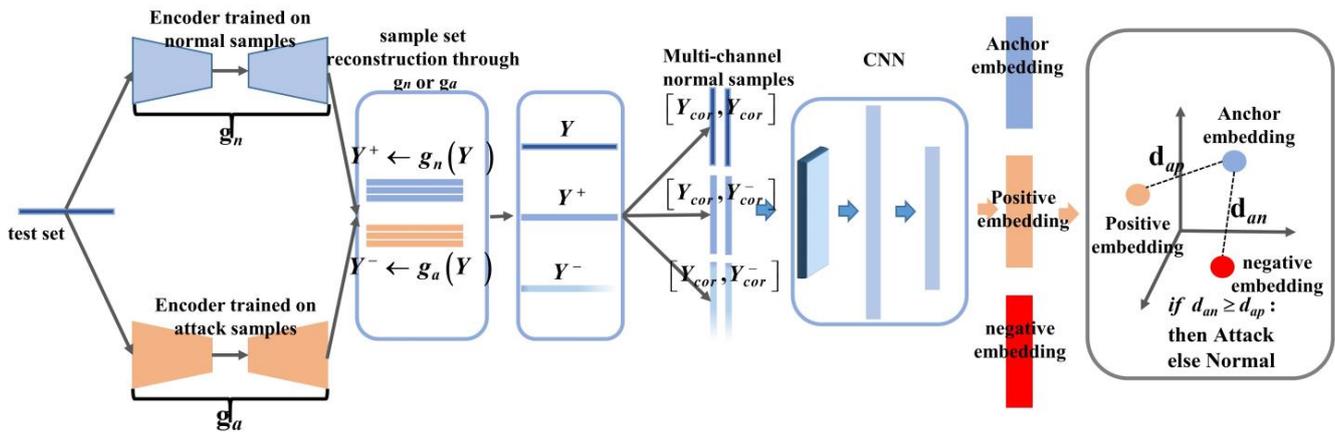


Figure 2. MCLDM prediction stage.

MCLDM prediction stage is described in Algorithm 2. Now, we consider a sample that needs to be queried  $Y \subset \mathbb{R}^D$ . In the first step, the data are reconstructed by the normal autoencoder  $g_n$  and attack autoencoder  $g_a$ . In the second step, multi-channel data construction is implemented. In the third step, the embedding representation of the output of the TCNN is calculated to achieve the judgment, as shown in Equations (2) and (3).

Finally, compare the distance between  $Dis_n$  and  $Dis_a$ . If  $Dis_n < Dis_a$ , classify the label of  $X$  as a normal network data stream, otherwise classify the label of  $X$  as an attack network data stream.

---

#### Algorithm 2 MCLDM prediction stage

---

**Input:**  $D$ : query sample set  $X \subset \mathbb{R}^D$ , as well Triplet Convolutional Neural Network  $\Phi$

**Output:** (*label*): the predicted label

- 1 **Begin:**
  - 2      $X^+ = g_n(X)$   $X^- = g_a(X)$
  - 3      $[X_{cor}, X_{cor}^+, X_{cor}^-] \leftarrow [X, X^+, X^-]$
  - 4      $Anchor = \Phi([X_{cor}, X_{cor}])$   $positive = \Phi([X_{cor}, X_{cor}^+])$
  - 5      $negative = \Phi([X_{cor}, X_{cor}^-])$
  - 6      $Dis_n = \text{Euclidean\_distance}(anchor, positive)$
  - 7      $Dis_a = \text{Euclidean\_distance}(anchor, negative)$
  - 8     If  $Dis_n < Dis_a$ :
  - 9         1 |  $label = normal$
  - 10     Else:
  - 11         2 |  $label = attack$
  - 12 **Return label**
-

#### 4. Implementation Details

MCLDM is implemented in python 3.8, and the framework used for the deep neural network is TensorFlow 2.8 with Keras 2.8., where the API for data preprocessing includes Scikit-learn. For training with the dataset, we used the library structure Parzen estimator algorithm implemented in the Hyperopt library for automatic tuning; this way 20% of the dataset is used as the test set, and in particular, our procedure used random sampling to select the validation set. The hyperparameter values for the automatic search using the tree-structured Parzen estimation are shown in Table 2. The cpu of the device we use is as follows: 15 cores/GPU, Intel®Xeon(R) Platinum 8358P CPU @ 2.60 GHz. The GPU model is RTX3080 with 10 GB of video memory. The system of the device is ubuntu20.08.

**Table 2.** Hyperparameter Value.

Auto-Tuning Parameter Names	Autoencoders	TCNN
batchsize	[25, 26, 27, 28, 29]	[25, 26, 27, 28, 29]
lr	[0.0001, 0.1]	[0.0001, 0.1]
Dropout	[0, 1]	[0, 1]

Each autoencoder includes three fully connected layers with 32, 16, and 32 neurons, and two dropout layers to prevent overfitting of the neuronal network. The neurons in each layer use Relu as the activation function and speed up the training of the network.

The TCNN is composed of three AlexNet and fully connected layers with shared weights. Each convolutional neural network is a deep neural network consisting of five convolutional layers, three pooling layers, one Flatten layer, three fully connected layers, and two Dropout layers. The activation function of the first two fully connected layers is Relu to speed up the network training, and the activation function of the final embedding representation layer is Sigmoid, which aims to make the size of the output embedding representation of [0, 1] in each dimension. Finally, the Euclidean distance of the output embedding is calculated and the data are determined as normal or attack samples by the distance.

#### 5. Experimental Validation

For the experimental evaluation, we use two benchmark datasets, CICIDS17 and KDDCUP99, which have a long timespan and are sufficient to validate the network intrusion detection to which MCLDM is applicable.

##### 5.1. Dataset Description

CICIDS17 was collected by the Canadian Institute of Cyber Security in 2017 and contains a total of 5 days of network intrusion logs collected from 3 July 2017 to 7 July 2017, where each network traffic sample includes 79 characteristics of information. The dataset includes eight data types, one normal sample and seven attack samples. The data contain 100,000 training sets and 900,000 test sets. The amount of normal traffic is much larger than the amount of attack data (80% vs. 20%) in both the training and test sets.

The KDDCUP99 dataset was adopted in the KDD competition in 1999, and has been frequently used to evaluate network intrusion-detection models since then. The dataset consists of five data types, one normal sample and four attack samples. The dataset consists of 494,021 training samples and 31,029 test samples. The amount of normal traffic is much smaller than the amount of attack data (19.7% vs. 80.5%) in both the training and test sets. The specific database description is shown in Table 3. In the dataset description we labeled the imbalance of the dataset. The data are unbalanced during training and testing.

**Table 3.** Description of the data set.

		Dataset	
		CICIDS17	KDDCUP99
Attributes	Total	79	42
	Total	100,000	494,021
Training set	Normal flows	80,000 (80%)	97,278 (19.7%)
	Attacking flows	20,000 (20%)	396,743 (80.3%)
	Total	900,000	31,1029
Testing set	Normal flows	720,000 (80%)	60,593 (19.5%)
	Attacking flows	180,000 (20%)	250,436 (80.5%)

Using two datasets for the training stage, we select the training set as the training sample and the test sample by random stratified sampling, where the training sample is 80% of the training set and the test sample is 20% of the training set. In the laboratory, we use data normalization. The purpose of normalizing the data is to reduce the training time of MCLDM and to accelerate the convergence of MCLDM. After hot-coding the data, we use the min-max scalar technique in Scikit-learn, with Equation (4).

$$X_{\text{std}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \quad (4)$$

where  $X_{\text{std}}$  denotes the representation of the input data sample  $X$  after normalization,  $X_{\min}$  denotes the minimum amount of some feature value, and  $X_{\max}$  denotes the maximum amount of some feature value.

### 5.2. Evaluation Metrics

Accuracy, Precision, Recall, and F1-Score metrics to evaluate the performance of the proposed MCLDM are calculated as in Table 4. All results are calculated from four variables: TN denotes the number of correctly predicted normal samples, TP denotes the number of correctly predicted attack samples, FN denotes the number of incorrectly predicted normal samples, and FP denotes the number of incorrectly predicted attack number samples.

**Table 4.** Evaluation metrics.

Metric	Mathematical Formulae
Accuracy	Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$
Precision	Precision = $\frac{TP}{TP+FP}$
Recall	Recall = $\frac{TP}{TP+FN}$
F1-Score	F1-Score = $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

This section may be divided by subheadings. It should provide a concise and precise description of the experimental results, their interpretation, as well as the experimental conclusions that can be drawn.

### 5.3. Observation Model Using Performance Metrics

We validated the MCLDM using Accuracy, Precision, Recall, and F1-Score evaluation metrics separately, as shown in Figure 3, and MCLDM performed better on each evaluation metric. The performance of Accuracy, Recall, and F1-Score on data set CICIDS17 is better than that on data set KDDCUP99, and the performance of Precision on data set KDDCUP99 is better than that on CICIDS17. Precision was 99.69%, Recall was 92.69%, and F1-Score was 96.06%. In the data set KDDCUP99, the Accuracy was 98.43%, Precision was 98.65%, Recall was 97.17%, and F1-Score was 97.93%. We measured the detection time of each sample by adjusting the size of the test CICIDS and KDDCUP99 datasets separately, and we found that changing the size of the data did not affect the test results of the samples.

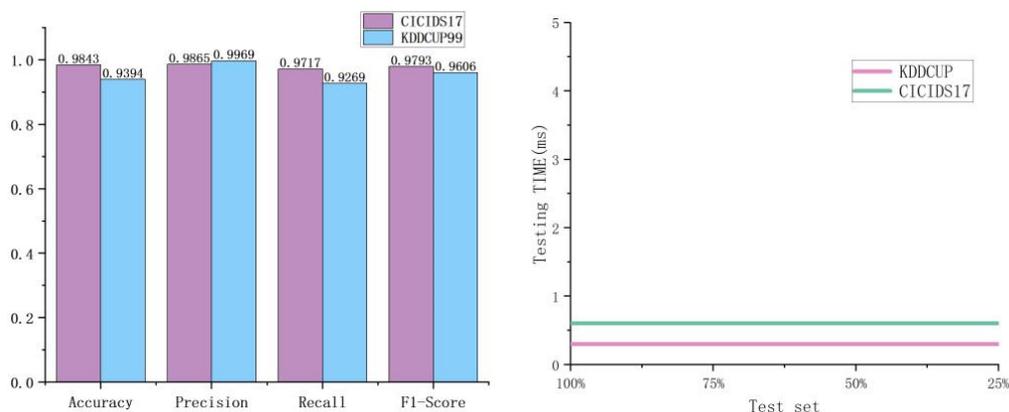


Figure 3. Performance evaluation metrics for CICIDS17 and KDDCUP99 datasets.

### 5.3.1. Model Accuracy and Loss Variation

The Accuracy and loss of our proposed MCLDM during the training stage are plotted in Figure 4 above, from which can be seen the fact that MCLDM changes very rapidly with increasing epochs at the beginning of the training set, indicating that the proposed MCLDM has good learning performance. As the epoch increases, the changes in Accuracy and loss level off gradually. When the Accuracy and loss curves of MCLDM start to deviate from the level of flatness, we stop the training of MCLDM.

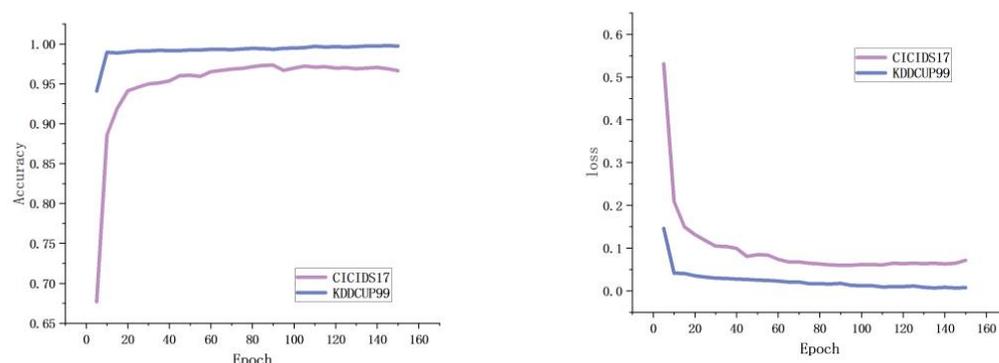
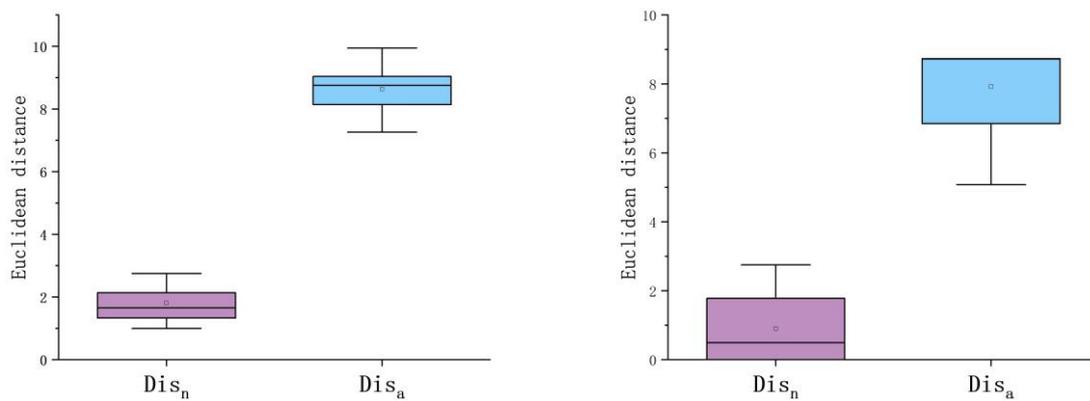


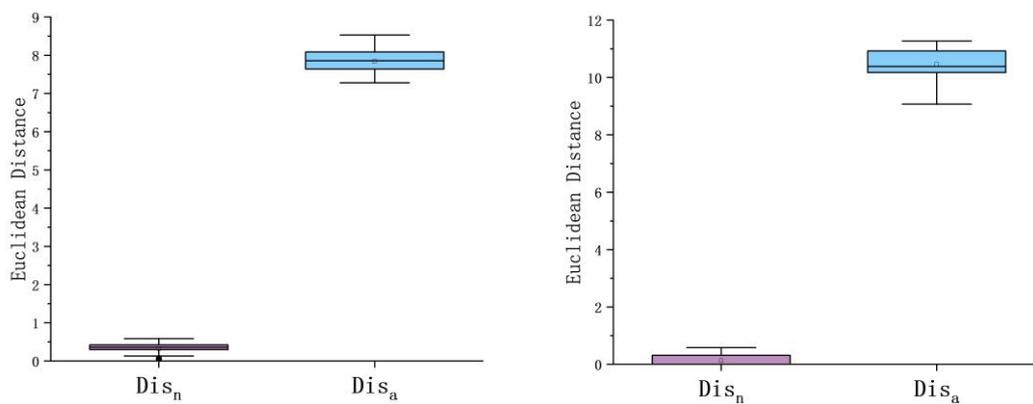
Figure 4. Accuracy and loss variation when models are trained using CICIDS17, KDDCUP99 dataset.

### 5.3.2. Embedding Analysis

We analyze the embeddings of MCLDM output to show how MCLDM performs contrastive learning to distinguish normal network flows from attack network flows. MCLDM is made easier to distinguish between normal and attack data streams by decreasing the Euclidean distance between the anchor point and the positive sample embedding representation, i.e., decreasing  $Dis_n$ , while increasing the Euclidean distance between the anchor point and the negative sample, i.e., increasing  $Dis_a$ . Figures 5 and 6 show the difference in the values of  $Dis_n$  and  $Dis_a$  during training and testing in datasets CICIDS17 and KDDCUP99, respectively. By analyzing Figures 4 and 5, it is obvious that the distance between the anchor embedding and the positive sample embedding is much smaller than that between the anchor embedding and the negative sample embedding. By comparing the embedding representation analysis of the training and test sets, it is obvious that both achieve the expected results.



**Figure 5.** Embedding analysis (on the CICIDS17 dataset) is shown on the left on the training set and on the right on the test set.



**Figure 6.** Embedding analysis (on the KDDCUP99 dataset) is shown on the left on the training set and on the right on the test set.

### 5.3.3. Ablation Study

Our research continues to analyze the following reasons that together contribute to the good accuracy of MCLDM in network intrusion detection.

- (1) Data reconstruction and synthesis of additional information are achieved through an autoencoder.
- (2) The multi-channel representation enriches the features of the raw data and increases the gap between different classes of data.
- (3) TCNN can contribute to the accuracy of MCLDM.

For this purpose, we use four architectural configurations as a baseline, which are defined by removing autoencoder, multichannel data construction, and convolutional neural network from MCLDM. We have considered the following four architectures.

- (a) NN: It is fully connected by the last three layers of the MCLDM architecture.
- (b) ANN: It is composed of an autoencoder and NN structure. This structure contains an autoencoder and can explain the advantages of using an autoencoder.
- (c) CNN: It is composed of CNN structure in MCLDM structure, and its principle is similar to NN. It is required to convert the input sample  $[X]$  to  $[X_{cor}]$  before training the model.
- (d) ACNN: It is a structure similar to CNN, but differs from the MCLDM structure in that it lacks the step of multi-channel data reconstruction.

We verified the performance of MCLDM, NN, ANN, CNN, and ACNN in datasets CICIDS17 and KDDCUP99, respectively. In the Table 5, we show the results of Accuracy and F1-score in different datasets, respectively, where MCLDM outperforms the other

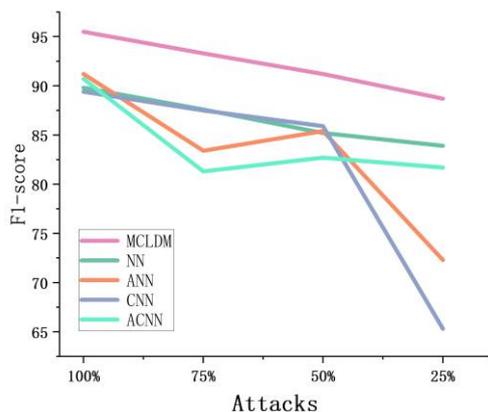
structures. This also proves that the combination of autoencoder, multichannel data reconstruction, and convolutional neural network is beneficial to obtain better accuracy in network intrusion detection.

**Table 5.** Accuracy and F1-score of MCLDM, NN, ANN, CNN, ACNN architecture validated in datasets CICIDS17 and KDDCUP99.

Dataset		Architecture				
		MCLDM	NN	ANN	CNN	ACNN
CICIDS17	Accuracy	98.43	95.89	96.70	95.69	96.50
	F1-score	97.93	89.53	91.72	89.32	91.31
KDDCUP99	Accuracy	93.94	92.02	91.92	92.10	92.11
	F1-score	96.06	94.80	94.75	94.92	94.80

### 5.3.4. Data Imbalance Verification

To verify that MCLDM remains robust in the data type imbalance phenomenon, we performed a validation using dataset CICIDS17, that was collected in a realistic network scenario that includes data-type imbalance sites, including 20% of attack data and 80% of normal data. We performed the validation by adjusting the amount of attack data in the dataset, and these experimental data include all normal data and different percentages of attack samples. We conducted four experiments using 100%, 75%, 50%, and 25% attack samples, respectively. Different degrees of data imbalance scenarios are achieved by reducing the use of attack samples for training. By experimenting with MCLDM, NN, ANN, CNN, and ACNN structures, the F1-score changes as shown in Figure 7. We find that the F1-score decreases for all algorithms, but MCLDM continues to outperform the other structures for different degrees of data imbalance, which also proves that MCLDM can handle data imbalance scenarios.



**Figure 7.** F1-score values of MCLDM, NN, ANN, CNN, and ACNN with the number of attacks in the CICIDS2017 dataset.

### 5.3.5. Comparison with Competitive Rivals

Our experiments are trained and predicted on CICIDS17, KDDCUP99. We compare the proposed MCLDM with competing adversaries. We compare the performance of Accuracy and F1-score metrics with competing lines in the recent state-of-the-art literature, respectively.

The Accuracy and F1-Score of MCLDM in all datasets are reported in Table 3, and a comparison of the data in Table 6 shows that good results were obtained for both Accuracy and F1-score metrics of MCLDM in datasets CICIDS17 and KDDCUP99. In the literature 24, good experimental results were also obtained using AE combined with deep metric learning (DML), where DML is the paradigm of contrastive learning, respectively. The contrastive analysis shows that MCLDM outperforms the latest competing model on dataset CICIDS17 by 0.19% for the Accuracy metric and 2.23% for the F1-score metric than the latest competing

model. MCLDM exceeds the Accuracy metric of the latest competitive model by 0.44% on the dataset KDDCUP99, and the F1-score is 0.26% higher than the latest competitive model.

**Table 6.** Accuracy and F1-cores tested at CICIDS17 and KDDCUP99.

Dataset	Algorithm	Description	Accuracy	F1-Score
CICIDS17	MCLDM	AE + TCNN	98.43	97.93
	RENOIR [22]	AE + DML	98.24	95.70
	MINDFUL [23]	AE + CNN	97.90	94.93
	THEODORA [25]	AE + CNN	98.03	95.25
KDDCUP99	MCLDM	AE + TCNN	93.94	96.06
	RENOIR [22]	AE + DML	93.50	95.80
	MINDFUL [23]	AE + CNN	92.49	95.13
	AE-LSTM [26]	AE + LSTM	90.5	91
	THEODORA [25]	AE + CNN	92.97	95.46

## 6. Conclusions

In this study, we propose a novel model for network intrusion detection, which uses autoencoders to reconstruct and noise-reduce the raw one-dimensional network stream data; calculates the cross-correlation matrix of the reconstructed data; combines the cross-correlation matrix to obtain multi-channel data; and, finally, uses a TCNN to achieve feature extraction of multi-channel data. Contrastive learning is achieved according to the prescribed loss function to achieve the training effect and show a good accuracy in the test set. The main idea is that after noise reduction, the data are combined into multi-channel data by autocorrelation calculation, so that the feature difference between different types of data will increase, which helps to distinguish the normal network flow from the abnormal network flow. By comparing with other techniques, we are good at dealing with the data imbalance problem, we used TCNN and finally designed MCLDM. MCLDM has some limitations, and the model is to transform the network intrusion-detection problem into a binary classification problem, and does not implement multiple classifications of network intrusions; therefore, the specific classes of network intrusions are not known, so our next task will be to implement network into specific classifications.

We evaluated the effectiveness of MCLDM using two benchmark datasets. The experimental analysis fully demonstrates the effectiveness of our proposed MCLDM approach. In particular, it has a good performance in dealing with the problem when the data are unbalanced. It is experimentally demonstrated that the multi-channel data constructed by class-specific autoencoders and using unsupervised contrastive learning helps to separate the different classes of data and finally achieve network intrusion detection.

**Author Contributions:** Methodology, J.L.; Validation, Y.Z.; Investigation, Y.X.; Resources, X.G. and B.S.; Funding acquisition, Y.W. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** Data available on request due to restrictions of privacy or ethical.

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

## References

1. Huang, Y.L.; Hung, C.Y.; Hu, H.T. A Protocol-based Intrusion Detection System using Dual Autoencoders. In Proceedings of the 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS), Hainan, China, 6–10 December 2021; pp. 749–758. [\[CrossRef\]](#)
2. Patil, A.P.; Premkumar, H.; Kiran, M.H.M.; Hegde, P. JARVIS: An Intelligent Network Intrusion Detection and Prevention System. In Proceedings of the 2022 IEEE Fourth International Conference on Advances in Electronics, Computers and Communications (ICAIECC), Bengaluru, India, 10–11 January 2022; pp. 1–6. [\[CrossRef\]](#)
3. Oh, S.H.; Yu, X.; Stefanie, J.; Silvio, S. Deep metric learning via lifted structured feature embedding. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 4004–4012.
4. Najafabadi, M.; Villanustre, F.; Khoshgoftaar, T.; Seliya, N.; Wald, R.; Muharemagic, E. Deep learning applications and challenges in big data analytics. *J. Big Data* **2015**, *2*, 1. [\[CrossRef\]](#)
5. Dong, B.; Wang, X. Comparison deep learning method to traditional methods using for network intrusion detection. In Proceedings of the 8th IEEE International Conference on Communication Software and Networks (ICCSN), Beijing, China, 4–6 June 2016; pp. 581–585.
6. Shone, N.; Ngoc, T.; Phai, V.; Shi, Q. A deep learning approach to network intrusion detection. *IEEE Trans. Emerg. Top. Comput. Intell.* **2018**, *2*, 41–50. [\[CrossRef\]](#)
7. Zheng, J.; Peng, L. An Autoencoder-Based Image Reconstruction for Electrical Capacitance Tomography. *IEEE Sens. J.* **2018**, *18*, 5464–5474. [\[CrossRef\]](#)
8. Johnson, J.; Khoshgoftaar, T. Survey on deep learning with class imbalance. *J. Big Data* **2019**, *6*, 1–54.
9. Hoffer, E.; Ailon, N. Deep metric learning using triplet network. In *Similarity-Based Pattern Recognition*; Feragen, A., Pelillo, M., Loog, M., Eds.; Springer International Publishing: Cham, Switzerland, 2015; pp. 84–92.
10. Hermans, A.; Beyler, L.; Leibe, B. Defense of the Triplet Loss for Person Re-Identification. *arXiv* **2017**, arXiv:1703.07737.
11. Huang, S.; Lei, K. Igan-ids: An imbalanced generative adversarial network towards intrusion detection system in ad-hoc networks. *Ad Hoc Networks* **2020**, *105*, 102177. [\[CrossRef\]](#)
12. Yıldırım, Ö.; Baloglu, U.B.; Acharya, U.R. A deep convolutional neural network model for automated identification of abnormal EEG signals. *Neural Comput. Appl.* **2018**, *32*, 15857–15868. [\[CrossRef\]](#)
13. Araki, S.; Hayashi, T.; Delcroix, M.; Fujimoto, M.; Takeda, K.; Nakatani, T. Exploring multi-channel features for denoising-autoencoder-based speech enhancement. In Proceedings of the 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), South Brisbane, QLD, Australia, 19–24 April 2015; pp. 116–120.
14. Dashtipour, K.; Gogate, M.; Adeel, A.; Ieracitano, C.; Larjani, H.; Hussain, A. Exploiting deep learning for Persian sentiment analysis. In *International Conference on Brain Inspired Cognitive Systems*; Springer: Cham, Switzerland, 2018; pp. 597–604.
15. Liu, L.; Wang, P.; Lin, J.; Liu, L. Intrusion detection of imbalanced network traffic based on machine learning and deep learning. *IEEE Access* **2021**, *9*, 7550–7563. [\[CrossRef\]](#)
16. Zhou, Y.; Cheng, G.; Jiang, S.; Dai, M. Building an efficient intrusion detection system based on feature selection and ensemble classifier. *Comput. Netw.* **2020**, *174*, 107247. [\[CrossRef\]](#)
17. Sun, X.; Lv, M. Facial expression recognition based on a hybrid model combining deep and shallow features. *Cogn. Comput.* **2019**, *11*, 587–597. [\[CrossRef\]](#)
18. Zeng, Y.; Gu, H.; Wei, W.; Guo, Y. Deep–Full–Range: A Deep Learning Based Network Encrypted Traffic Classification and Intrusion Detection Framework. *IEEE Access* **2019**, *7*, 45182–45190. [\[CrossRef\]](#)
19. Althubiti, S.A.; Jones, E.M.; Roy, K. LSTM for anomaly-based network intrusion detection. In Proceedings of the 2018 28th International Telecommunication Networks and Applications Conference (ITNAC), Sydney, NSW, Australia, 21–23 November 2018; pp. 1–3.
20. Lopez-Martin, M.; Carro, B.; Sanchez-Esguevillas, A.; Lloret, J. Shallow neural network with kernel approximation for prediction problems in highly demanding data networks. *Expert Syst. Appl.* **2019**, *124*, 196–208. [\[CrossRef\]](#)
21. Kaya, M.; Hasan, Ş. Deep metric learning: A survey. *Symmetry* **2019**, *11*, 1066. [\[CrossRef\]](#)
22. Andresini, G.; Appice, A.; Mauro, N.D.; Loglisci, C.; Malerba, D. Multi-Channel Deep Feature Learning for Intrusion Detection. *IEEE Access* **2020**, *8*, 53346–53359. [\[CrossRef\]](#)
23. Andresini, G.; Annalisa, A.; Donato, M. Autoencoder-based deep metric learning for network intrusion detection. *Inf. Sci.* **2021**, *569*, 706–727. [\[CrossRef\]](#)
24. Ma, Y.; Jinglin, S.; Jinlong, H. Reconfigurable remote radio head design and implementation for super base station applications. *Ann. Telecommun.* **2018**, *73*, 639–650. [\[CrossRef\]](#)
25. Maleh, Y.; Shojafar, M.; Alazab, M.; Baddi, Y. (Eds.) *Machine Intelligence and Big Data Analytics for Cybersecurity Applications*; Springer: Cham, Switzerland, 2021.
26. Mushtaq, E.; Zameer, A.; Umer, M.; Abbasi, A.A. A two-stage intrusion detection system with auto-encoder and LSTMs. *Appl. Soft Comput.* **2022**, *121*, 108768. [\[CrossRef\]](#)

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.