A Multiple-Layer Representation Learning Model for Network-Based Attack Detection

XUEQIN ZHANG[1], JIAHAO CHEN [1], YUE ZHOU[1],
LIANGXIU HAN [2], (Member, IEEE), AND JIAJUN LIN[1]
[1]College of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China
[2]School of Computing, Mathematics and Digital Technology, Manchester Metropolitan University, Manchester M15 6BH, U.K.

Corresponding author: Xueqin Zhang (zxq@ecust.edu.cn)

ABSTRACT Accurate detection of network-based attacks is crucial to prevent security breaches of information systems. The recent application of deep learning approaches for network intrusion detection has shown promising. However, the challenges remain on how to deal with imbalance data and small samples as well as reducing false alarm rate (FAR). To address these issues, this work has proposed a multiple-layer representation learning model for accurate end-to-end network intrusion detection by combining deep convolutional neural networks (CNN) with gcForest. The contributions of this work lie in 1) a new data encoding scheme based on P-Zigzag to encode network traffic data into two-dimensional gray-scale images for representation learning without loss of original information; 2) The combination of gcForest and CNN allows accurate detection on imbalanced data and small scale data with fewer hyperparamters comparing to most existing deep learning models, which increase computational efficiency. The proposed approach is based on a multiple-layer approach consisting of a coarse layer and a fine layer, in which the coarse layer with the improved CNN model (GoogLeNetNP) focuses on identification of N abnormal classes and a normal class. While in the fine layer, an improved model based on gcForest (caXGBoost) further classifies the abnormal classes into N-1 subclasses. This ensures fine-grained detection of various attacks. The proposed framework has been compared with the existing deep learning models using three real datasets (a new dataset NBC, a combination of UNSW-NB15 and CICIDS2017 consisting of 101 classes). The experimental results show that our proposed method outperforms other single deep learning methods (i.e., AlexNet, VGG19, GoogleNet, InceptionV3, ResNet18) in terms of accuracy, detection rate, and FAR, which demonstrates its effectiveness in detecting fine-grained attacks and handling imbalanced datasets with high-precision and low FAR.

INDEX TERMS Network intrusion detection, convolutional neural networks, deep random forests, representation learning.

I. INTRODUCTION

Network security attacks have become an increasingly serious global problem and the endless cybersecurity accidents are a major threat to social and economic development. Intrusion detection technology is a security mechanism that dynamically monitors, prevents, and defends against network and system intrusion. It is an important component of a network security defense system. It can detect and report unauthorized operations or anomalies in the system. The traditional machine learning based methods for intrusion detection can be broadly divided into two main steps: feature extraction and selection and classification [1]. The classical

feature extraction and selection algorithms such as Principal Component Analysis (PCA) [2], Correlation-based Feature Selection method (CFS) [3] etc. were proposed to extract most discriminative features, while the traditional classification models have Support Vector Machine (SVM) [4], Neural Network [5], Naive Bayes and Decision Trees [6] etc. were applied for classification of network attacks. However, one of the limitations faced by traditional machine learning approaches in intrusion detection is that the classification mainly relies on features selected based on preprocessing algorithms rather than the entire raw data [7]. This may lead to inaccurate intrusion detection due to information loss and incomplete feature extraction during the feature selection step. Deep learning is a representative of machine learning algorithms in recent years and has achieved great success in

the field of image recognition and speech recognition [8], [9]. One of the reasons for the breakthrough is that deep learning can automatically learn the nonlinear correlation and extract features from the raw data to achieve the end-to-end learning [10]. Such advantages attract an increasing number of scholars in the security field to pursue deep learning-based solutions. For instance, Mostafa et al. [11] combined Deep Belief Network (DBN) with intrusion detection technology to achieve higher detection accuracy than the ones based on SVM. Besides, Long short-term memory (LSTM), Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN) are also applied in intrusion detection [12], [13].

Despite the existing works are encouraging, there are still challenges in exploring deep learning approaches for network intrusion detection. First, the deep learning relies on a large number of samples for model training. It is still a challenging task for training a model on a small-scale dataset. Moreover, the unbalanced data is a generic problem for deep learning. The unbalanced data will affect the performance of the model, leading to the high false alarm rate and high false miss rate of some certain classes with fewer samples. Quamar et al. [14] evaluated deep learning models such as self-taught learning(STL) on NSL-KDD, among which U2R attack (user to root) have fewer samples and the classification performance is low.

To address the challenges mentioned above, in this work, we have proposed a new intrusion detection method by combining CNN with deep gcForest. The contributions of this work include:

1) a novel encoding method based on P-Zigzag has been proposed which can encode the raw network data into gray-scaled images for representation learning without information loss.

2) a multiple layer approach where the coarse layer of an improved CNN network based on GoogLeNet (called GoogLeNetNP) can identify N abnormal classes and a normal class. While in the fine layer, an improved gcForest (caXGBoost) provides N-1 gcForest models for more subclasses. This ensures fine-grained detection of various attacks.

The rest of this paper is organized as follows. Section II describes the related work. Section III presents the design and implementation of the proposed method. Section IV mainly covers the evaluation methodology and experimental results, and Section V concludes the work and highlight future works.

II. RELATED WORK
A. MACHINE LEARNING BASED INTRUSION DETECTION METHODS
Numerous researches based on machine learning have been proposed for intrusion detection. Lee et al. [15] proposed a data mining framework to learn rules that accurately identified the behavior of intrusions and normal activities. Kruegel et al. [16] proposed an event classification scheme based on Bayesian networks, which improved the aggregation of different model outputs, allowing

one to seamlessly incorporate additional information. Mukkamal et al. [17] applied neural networks and support vector machines in intrusion detection to discover useful patterns. However, these traditional methods are hard to solve the high false alarm rate and missed alarm rate. Especially, with the increasing data size and new attacks, accurate and timely detection becomes more challenging.

As a branch of machine learning, deep learning-based approaches have been also applied for intrusion detection due to its successful applications in various domains such as object detection [18], natural language processing [19]. Shone et al. [20] proposed an asymmetric depth autoencoder with two layers of NDAE stacked in the detector, using a random forest to perform classification tasks at the end of the network with an average accuracy of 97.85% on KDD99, but fail to perform classification on small samples. Fiore et al. [21] proposed a semi-supervised learning method based on discrete-confined Boltzmann machines and achieved better performance than the one based on DBN. However, this method had a difficulty in dealing with the unbalanced data and small samples. Potluri et al. [22] used DBN for feature extraction, and softmax and SVM for classification at the end of the network. The detection accuracy of categories of attacks on the NSL-KDD dataset was 80-90%, while that of small categories was about 20%. Yadav et al. [23] proposed a heap auto-encoder to detect DDoS attacks. The experimental analysis using AL-DDoS dataset with 8 feature dimensions showed that the algorithm could identify the attacks with an average detection rate of 98.99%. However, this method is limited in DDoS detection with only three types of attacks (i.e. request flooding, session flooding, and asymmetric attack). In 2017, Yin et al. [24] proposed the RNN cyclic neural network for intrusion detection based on NSL-KDD. The accuracy, false negative rate and false positive rate are superior to traditional machine learning methods. Yuan et al. [25] used LSTM to perform time series classifier experiments on the ISCX2012 dataset by adding a sliding window on the dataset to get the three-dimensional data for classifier training. Compared with GRU, CNNLSTM and other algorithms, LSTM is relatively better in terms of accuracy and recall rate. Li et al. [26] proposed to convert the 41-dimensional features of the NSL-KDD dataset into 8×8 images by encoding, which was trained by GoogLeNet [27] and ResNet50 [28]. The average detection rate of two test sets of NSL-KDD achieves 80%. Wang et al. [29] encoded the original network traffic of DARPA1998 [30] and ISCX2012 [31] directly by one-hot coding, and then used CNN and CNN-LSTM algorithms to perform classification respectively. Both studies demonstrated the effectiveness of the CNN in intrusion detection, but there is a room for improvement in fine-grained classification and unbalanced data.

B. DEEP RANDOM FOREST
gcForest (Multi-grained Cascade Forest) was put forward by Zhou et al. [32]. This method generates a deep forest

ensemble with a cascade structure, which enables gcForest to do representation learning.

Two main components of gcForest include the cascade structure and the multi-grained scanning. The former one is inspired by the representation learning in deep neural networks, which mostly relies on the layer-by-layer processing of raw features. Each level is an ensemble of decision tree forests. The latter one uses sliding windows to scan raw data, which can further enhance representational learning ability by multi-grained scanning when the inputs are high dimensional (like convolution in CNN), potentially enabling gcForest to be contextual or structural aware. gcForest can include different types of forests to encourage the diversity, which is crucial for ensemble construction [33]. The original gcForest uses two completely random tree forests and two random forests [34]. Each completely-random tree forest contains 500 completely-random trees [35].

There are two main advantages of gcForest including 1) Comparing to deep neural networks, gcForest has far fewer hyper-parameters and its performance is quite robust [32]. 2) Forest classifiers perform better on imbalanced datasets and small samples. The main reasons are as follow: the number of cascade levels can be adaptively determined such that the model complexity can be automatically set, enabling gcForest to perform well on small-scale data. Besides, gcForest consists of units like random forests and other tree-related classifiers, whose advantages are to use ensemble learning to solve the imbalanced data. For instance, random forests, a typical kind of ensemble learning, are not sensitive to multicollinearity, leading to the robustness on missing and imbalanced data and can predict well with up to several thousand explanatory variables [36].

Inspired by gcForest, more deep forest structures are proposed. Miller et al. [37] proposed FTDRF (Forward Thinking Deep Random Forest), which resembles gcForest. The only different part is the parameters sent between layers. Zhou et al. also proposed two other structures to handle different problems, eForest (EncoderForest) [38] for back propagation decoding and mGBDTs (multi-layered GBDT forests) [39] for non-differentiable problem.

So far, gcForest has been applied to different domains for image processing. Li et al. [40] used gcForest for clothes classification. Ma et al. [41] combined gcForest and multi-scale fusion for change detection based on SAR images. Moreover, Liu et al. [42] utilized the advantage of gcForest on dealing with small sample datasets to diagnose the faults of rolling bearing, as the labeled image samples of rolling bearing are few. In this work, for the first time, we extend gcForest by adding XGBoost as the unit and apply it to intrusion detection.

III. THE PROPOSED DETECTION MODEL (DCF-IDS)
A. OVERVIEW OF MULTIPLE-LAYER REPRESENTATION LEARNING
To overcome the limitations in dealing with small samples and unbalanced data, we have proposed a multiple-layer
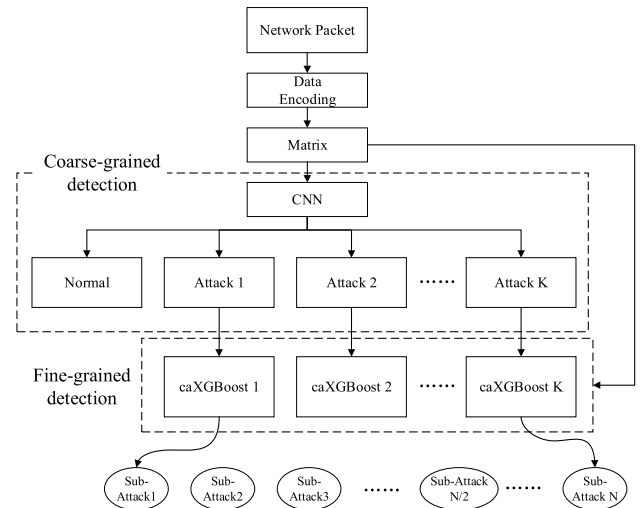
FIGURE 1. The structure of the proposed multi-layer representation learning model (DCF-IDS).

framework for accurate intrusion detection, consisting of two layers. The first layer is used for coarse-grained detection based on the GoogLeNetNP, which can identify normal traffic and the coarse-grained attack classes and reduce the false miss rate. The second layer is used for fine-grained detection. To address the issues with the imbalanced data and fewer samples, the improved deep random forest model is used to identify attack subclasses and improve detection accuracy.

The basic model framework is shown in Figure 1. The network packet will be encoded into a two-dimensional image, the features of which are extracted by CNN and gcForest respectively. At the first layer, there will be only one CNN model to detect N classes, including normal class. At the second layer, there will be N-1 deep forest models for more subclasses. This model is named **DCF-IDS**.

B. A P-ZIGZAG ENCODING OF NETWORK TRAFFIC DATA FOR REPRESENATIVE LEARNING
We have proposed a new data encoding scheme named 'P-Zigzag' to encode network traffic data two-dimensional gray-scale images for feeding the CNN model, which preserves the original information without loss.

Essentially, P-Zigzag scheme consists of three parts: data conversion, data padding and data encoding.

In the data conversion, the purpose is to convert data to decimal. The final form of the packet is expressed in hexadecimal on the physical layer. Since each bit information is represented by two hexadecimal numbers, we can convert the hexadecimal data into decimal data. Then the range of the value is 0-255, which is equivalent to the range of the image pixel value, and the data can be transformed into an image.

In the data padding, the purpose is to fill the data into the matrix as a pixel. The maximum data of each packet is 1518bit. In order to contain all the data part information, the matrix size can be defined as 64×64 for data storage. Since each packet size is different, the rest of the matrix

FIGURE 2. The structure of multi-layer representation learning models.

FIGURE 3. The result of P-Zigzag and IDCT.

FIGURE 4. The structure of coarse-grained detection models.

units will be padded with zero. Figure 2 shows an example of filling data with the form of zigzag, the direction of the arrow in Figure 2 represents the direction in which the data is filled. The purpose of such operation is that the data information can be concentrated in the low frequency part of the frequency domain, while the high frequency is 0, which resembles the result of image after discrete cosine transform (DCT) [43], that is, the gray-scale image is converted into DCT frequency domain map.

In the data encoding step, to feed the input to the representation learning model, we employ inverse discrete cosine transform (IDCT) to transform frequency domain map to the gray-scale image, as shown in Figure 3.

Each image has its own specific pattern texture that can be extracted features by CNN and deep forest models.

In summary, the caught packet data will be filtered to remove the header, and then the content of the data is transformed by P-Zigzag algorithm to two-dimensional image data.

C. COARSE-GRAINED DETECTION
As mentioned in Section II, with the support of large-scale training data, the CNN models can effectively detect anomalies and reduce the false miss rate. In this work, we have introduced a model called GoogLeNetNP (GoogLeNet for Network Packet), which was built on the GoogLeNet as the first layer classifier to implement coarse-grained detection and improve the model. GoogLeNetNP is a fine-tuning model structure for the specified attack detection based on Inception V2 by removing the BN (Batch Normalization) layer [44] and retaining Inception module.

The Inception structure uses the Network In Network (NIN) [45] idea to convolve simultaneously on multiple scales to extract features of different scales of the image. More feature maps mean more accuracy in the classification. The Inception structure has four branches. The first branch performs a 1×1 convolution on the input. The second branch uses a 1×1 convolution and then a 3×3 convolution. The third

branch uses a 1×1 convolution and then a 5×5 convolution. The fourth branch uses a 3×3 max-pooling and then uses a 1×1 convolution. Each branch introduces a 1×1 convolution kernel, and while lifting the convolution kernel receptive field, it also performs dimensionality reduction to speed up network calculation and reduce computational complexity, so that a layer of feature transformation can be added with a small amount of computation. The feature cascading layer connects the convolution results of $1 \times 1/3 \times 3/5 \times 5$, which prevents the demand for computing resources caused by the increase in the number of layers, and the width and depth of the network can be expanded. As shown in Figure 4, in GoogLeNetNP, two 3×3 convolution kernels are used to replace a 5×5 convolution to reduce the number of parameters and mitigate overfitting.

The BN layer plays an important role in many CNN-based classification tasks. BN handles the colorful image by normalizing the color distribution. In order to prevent the original contrast information from destruction, BN adds scale and shift parameters to offset the normalization effect and preserve the data nonlinear, but this increases the complexity and time of training. However, the normalization does not have much effect on the gray-scale image. Adding BN layer doesn't improve the performance of the gray-scale image classification, for instance, the slow training speed and poor detection accuracy. This is because the two-dimensional gray-scale image of the network data is a single-channel for the CNN model. Moreover, the matrix information after BN ignores the absolute difference between image pixels (or features), as the mean is zero and the variance is normalized. Only relative differences are considered. However, for network data images, each pixel is meaningful not redundant. The absolute difference between pixels and pixels is still very important. Therefore, in our case, BN layer has been removed.

The detailed structure of the improved model is shown in Table 1.

D. FINE-GRAINED DETECTION
To improve the accuracy, we have proposed to add a new layer (called a fine-grained layer) on top of the coarse-grained detection layer to further classify the undetected and/or

TABLE 1. The structure of the improved model based on GoogLeNet.

Type	Patch size / Stride
Convolution	$7 \times 7 / 2$
Max pool	$3 \times 3 / 2$
Convolution	$3 \times 3 / 1$
Max pool	$3 \times 3 / 2$
Inception(3a)	
Inception(3b)	
Max pool	$3 \times 3 / 2$
Inception(4a)	
Inception(4b)	
Inception(4c)	
Inception(4d)	
Inception(4e)	
Max pool	$3 \times 3 / 2$
Inception(5a)	
Avg pool	$2 \times 2 / 1$
Dropout(40%)	
linear	
softmax	

FIGURE 5. The structure of caXGBoost.

misclassified attacks from subclasses. However, with fine-grained classification, some subclasses may contain small samples and data may be imbalanced. While in practice, deep learning generally requires large-scale data for training models in order to achieve good detection results, the existing CNN-based models are not ideal for the fine-grained classification of attack subclasses. Inspired by the idea of gcForest where a deep forest ensemble with a cascade structure enables gcForest to train models on small sample and handle imbalanced dataset, we have proposed caXGBoost to solve the imbalanced datasets and sample size of the subclasses. caXGBoost is based on gcForest but only utilizes its cascade structure. The classification unit of caXGBoost adopts XGBoost [46] instead of random forests in the original gcForest. This is because XGBoost supports parallel programming without great loss of accuracy. In fact, XGBoost, namely eXtreme Gradient Boosting, is an integrated learning method that combines classification regression trees (CART trees). The XGBoost training process is an additional training. By optimizing the objective function in steps, the first tree is first optimized, and the second tree is optimized based on the optimization result of the first tree until the K tree is optimized. The objective function of XGBoost is as shown in (1).

$$obj(\theta) = \sum_{i}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 + C$$

$$(1)$$

where C is a constant, the first term is based on the loss function of the former $t - 1$ tree, y_i is the label of x_i, f_t represents t^{th} tree, $\hat{y}_i^{(t-1)}$ represents the prediction of the combination of t tree models, $l(y_i, \hat{y}_i)$ is the training loss of x_i. The latter two are regular terms, where T represents the number of leaf nodes of the tth tree. The larger γ and λ are, the greater the penalty value of T, which means the simpler model will be.
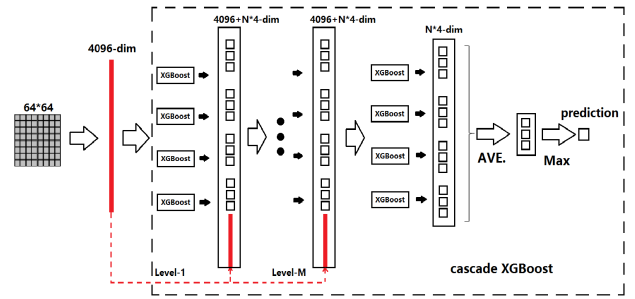
The second-order expansion of Taylor is performed on the above equation to obtain the final objective function of (2).

$$Obj^* = -\frac{1}{2} \sum_{j=1}^{T} w_j^* + \gamma T$$

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

$$(2)$$

Obj* indicates how good or bad the structure of the tree is. The smaller the value is, the better the structure will be. *Obj** is only related to T, regardless of the value of leaf node.

Moreover, using only one-unit classifier can simplify the model structure and reduce parameter dimensions to improve efficiency.

The structure of caXGBoost is shown in Figure 5. Suppose the input data is a 64×64 two-dimensional matrix, then it will be directly expanded into a 4096-dimensional vector as input data for training in a cascade network.

The output of each layer in the cascade structure is determined by the primitive input data vector V and the results of multiplying the number of forests N in each layer and the category k to be detected. Thus, the input and output vector dimensions between layers are as shown in (3).

$$Vector_{in(l)}^{out(l-1)} = m + k \times N$$

$$(3)$$

where m is the input data vector, $l - 1$ is the output layer of the previous layer, and l is the input layer of the current layer.

Supposing the cascade network consists of M layers forest structure, the unit classifier of each layer of forest is XGBoost. Then the input and output vector dimensions between layers will be $4096 + k \times N$. The output of the last layer is the vector result of $k \times N$. The prediction result is the maximum value of the average of the vector.

Compared with the gcForest, we removed the multi-grained scanning part for the simplicity and efficiency of the network without losing accuracy. The multi-grained scanning process is divided into two cases. When the input is a sequence vector, the length of the sampling window is set to K, the step size is *stride* and the padding is 0, then the number of subsamples is shown in (4).

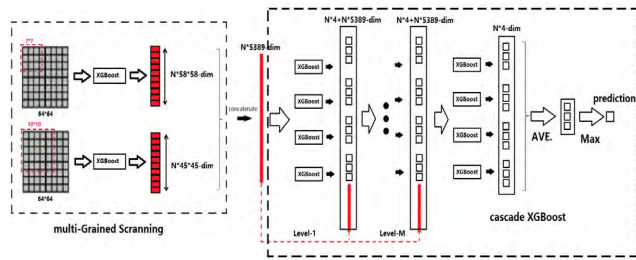$$S = \frac{m - K}{stride} + 1$$

$$(4)$$

FIGURE 6. The structure of caXGBoost.

where m is the input sequence data vector. We will get several K-dimensional sampling vectors, the number is S. When the input a two-dimensional matrix data, a rectangular sampling window of size $K \times K$ is set for sliding sampling, the step size is *stride* and there is still no filling operation. Suppose the input matrix is $m \times m$. Then the number of subsamples is shown in (5).

$$A = \frac{m - K}{stride} + 1$$
$$S = A^2 \tag{5}$$

where A is the result of one-side scanning. We will get S $K \times K$ sampling vectors. Suppose there are k classes to be detected, then the output of the multi-grained scanning will be $S \times k$, which will be sent to the cascaded part.

As shown in Figure 6, we let gcForest take XGBoost as the unit classifier for the comparison. Suppose the input data is a 64×64 two-dimensional matrix, the sliding window of 7×7 and 10×10 is used to acquire multiple feature information of the two-dimensional image through multi-granularity scanning, and the step is 1. The input image size is 64×64 and the number of classes to be detected is k. Vector of $k \times 58 \times 58$ and $k \times 45 \times 45$ can be obtained respectively with XGBoost. Combining the feature vectors of the multi-grained scanning yields an $k \times 5389$-dimensional vector as an input to the cascaded network. Then the input and output vector dimensions between layers will be $k \times 5389 + k \times N$, which will cost much more computation on the network. Besides, since the input data is fixed on 64×64 gray-scale image, the multi-grained scanning will generate redundant data affecting detection.

Thus, caXGBoost takes the advantage of gcForest on handling small samples and imbalanced data for the self-adaptivity and insensitive to multicollinearity, and has less hyperparameters than existing deep learning methods. Moreover, compared with gcForest, caXGBoost reduces the dimension of training data, which can improve the speed of training and have less effect on the accuracy of detection.

E. FLOW OF THE DCF-IDS FRAMEWORK
In DCF-IDS, we apply GoogLeNetNP as the first layer coarse-grained detection and apply caXGBoost as the second layer fine-grained detection. In summary, the pseudo-code of the DCF-IDS models is as follows:

DCF-IDS Algorithm

1. load GoogLeNetNP model
2. load caXGBoost model
3. get network traffic by tcpdump
4. transfer network data matrix by P-Zigzag
5. get batch matrix
6. use GoogLeNetNP to classify batch matrix and get results array
7. **for** index in results array:
8.   predict = the classification result of each network packet
9.   **if** predict == a certain attack class:
10.   sub_class_predict = corresponding_ caXGBoost_model.predict(batch_img[index]
11.   write log
12.   **else:**
13.   Normal network traffic

IV. EVALUATION AND DISCUSSION
A. EXPERIMENTAL METHODOLOGY
1) EXPERIMENTAL GOALS
The experimental goals are to evaluate the proposed framework DCF-IDS in the following aspects:

- Section B: evaluation of the multiple-layer models.
- Section C: evaluation of data encoding on feature extraction and dimension reconstruction.
- Section D: evaluation of the coarse-grained detection.
- Section E: evaluation of handling small samples and imbalance data.

Additionally, the proposed methods, including P-Zigzag, GoogLeNetNP and caXGBoost, will be tested respectively to verify their effectiveness and performance.

2) EVALUATION METRICS
Three metrics are used to evaluate the performance of the multi-cascaded representation learning models: accuracy, detection rate (DR) and FAR, which are commonly used in the field of intrusion detection. Accuracy is used to evaluate the overall of the system. DR is used to evaluate the system's performance with respect to its attack detection. FAR is used to evaluate misclassifications of normal traffic. Their definitions are presented below.

$$ACC = \frac{TP + TN}{TP + FP + FN + TN} \tag{6}$$

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

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

where TP is the number of instances correctly classified as X, TN is the number of instances correctly classified as Not-X, FP is the number of instances incorrectly classified as X, and FN is the number of instances incorrectly classified as Not-X.

TABLE 2. The general statistics of NBC.

Main Classes	Subclasses Number	Training	Test
Normal	/	246689	137248
DoS	21	35302	10887
Exploits	40	73414	28919
Fuzzers	10	24330	6986
Generic	5	9414	1866
Reconnaissance	7	2855	602
Shellcode	11	428	114
Virus	2	202	58
Webattack	4	8139	1338
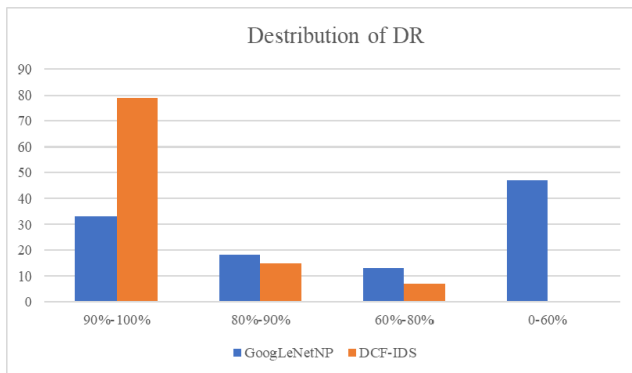
FIGURE 7. The distribution of DR between DCF-IDS and GoogLeNetNP.
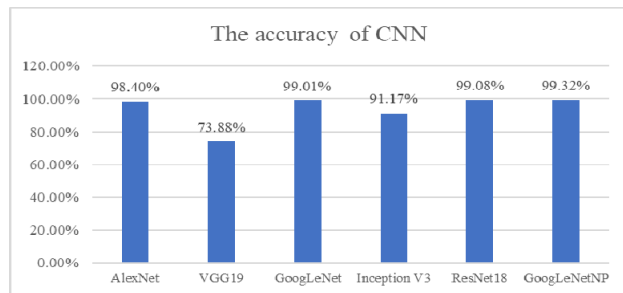
FIGURE 8. The accuracy of different CNN structures.

3) DATASETS

The dataset used in this paper consists of two open source datasets, including UNSW-NB15 (Australian Cyber Security Center dataset) [47], [48] and CIC-IDS2017 (Canadian Cyber Security Institute dataset) [49].

The UNSW-NB15 dataset includes contemporary attacks on real network activity and synthesis. There are nine types of attacks for simulated attacks, including Analysis, Backdoors, DoS, Exploits, Fuzzers, Generic, Reconnaissance, Shellcode, and Worms. According to the statistics provided by UNSW-NB15, the nine attack categories can be subdivided into 6,845 attack sub-categories, of which 3,068 are included in the CVE vulnerability database.

TABLE 3. The comparison between DCF-IDS and GoogLeNetNP on DoS.

	Classes / Subclasses	DCF-IDS DR(%)	DCF-IDS FAR(%)	GoogLeNetNP DR(%)	GoogLeNetNP FAR(%)
	Asterisk	98.69%	0.000%	97.37%	0.001%
	Browser	84.51%	0.004%	0.00%	0.027%
	DCERPC	100.00%	0.000%	0.00%	0.003%
	DNS	80.00%	0.001%	40.00%	0.002%
	FTP	92.86%	0.001%	35.71%	0.005%
	HTTP	99.58%	0.001%	98.21%	0.003%
	IIS Web Server	99.35%	0.001%	98.96%	0.001%
	IMAP	100.00%	0.000%	0.00%	0.001%
	LDAP	92.86%	0.000%	42.86%	0.002%
	Microsoft Office	94.97%	0.006%	70.15%	0.033%
DoS	Miscellaneous	99.68%	0.018%	98.76%	0.070%
	NetBIOS SMB	86.11%	0.002%	25.00%	0.012%
	RTSP	100.00%	0.000%	0.00%	0.001%
	SIP	100.00%	0.000%	25.00%	0.002%
	SMTP	93.33%	0.000%	14.29%	0.004%
	SNMP	80.00%	0.000%	0.00%	0.002%
	SSL	98.71%	0.000%	100.00%	0.000%
	TFTP	75.00%	0.000%	0.00%	0.001%
	Telnet	100.00%	0.000%	100.00%	0.000%
	VNC	83.33%	0.001%	50.00%	0.002%
	Windows Explorer	61.90%	0.002%	0.00%	0.005%

TABLE 4. The average DR and FAR between DCF-IDS and GoogLeNetNP.

Classes	DCF-IDS DR(%)	DCF-IDS FAR(%)	GoogLeNetNP DR(%)	GoogLeNetNP FAR(%)
Normal	99.88%	0.33%	99.93%	0.20%
DoS	91.47%	0.00%	42.68%	0.01%
Exploits	95.07%	0.02%	54.19%	0.02%
Fuzzers	99.26%	0.00%	98.55%	0.00%
Generic	96.05%	0.00%	84.08%	0.01%
Reconnaissance	98.83%	0.00%	69.21%	0.01%
Shellcode	89.46%	0.00%	33.75%	0.00%
Virus	82.05%	0.00%	67.02%	0.00%
Webattack	92.03%	0.00%	53.97%	0.01%

The lack of web-based attacks in UNSW-NB15 will be complimented by CIC-IDS2017 dataset, which includes both normal data and the latest web attack data. CIC-IDS2017 uses the B-Profile system proposed by Sharafaldin et al. [50] to describe the abstract behavior of human interaction and generate normal background traffic. Attacks implemented by datasets include brute force FTP, brute force SSH, DoS, Heartbleed, Web attack, infiltration, botnet and DDoS.

TABLE 5. The time cost among preprocessing methods.

	Converting Time(s)	Training Time(h)	Test Time(min)	Accuracy
P-Zigzag	86.76	3h	16.01	99.28%
OHE	730.00	17.35h	20.34	74.13%

TABLE 6. Testing results of preprocessing.

	P-Zigzag		OHE	
	DR	FAR	DR	FAR
Normal	99.88%	0.32%	76.87%	48.78%
DoS	98.28%	0.11%	83.11%	1.40%
Exploits	99.08%	0.17%	60.95%	8.52%
Fuzzers	97.52%	0.10%	67.15%	1.68%
Generic	95.03%	0.05%	52.52%	0.34%
Reconnaissance	97.06%	0.01%	5.65%	0.40%
Shellcode	87.85%	0.01%	14.04%	0.02%
Virus	72.00%	0.01%	0.00%	0.02%
Webattack	99.46%	0.00%	69.96%	0.14%

Although UNSW-NB15 and CIC-IDS2017 provide raw traffic data, the normal traffic and attack traffic in the PCAP file are mixed together. Therefore, the traffic data needs to be labeled. According to the attack records provided by the UNSW-NB15 and CIC-IDS2017 datasets, the calibration can be designed according to the characteristics of the original traffic and implemented by script code.

In order to meet the demand of the network detection on the raw traffic, this paper combine both datasets to form the new dataset called NBC, which has eight attack classes and the normal type. Each class also has subclasses, the total number of which is 100, as shown in Table 2. The total numbers of training set and testing set are 400773 and 188018 respectively. The detailed information and the numbers of training and test of each sub-classes are shown in AFFIX Table 13.

4) EXPERIMENTAL SETUP

In this research, we have used the most current deep learning frameworks, Caffe [51] and Tensorflow [52], which are run on the Ubuntu 18.04 64-bit OS. The experiment is performed on a server DELL T630 with 32GB of memory and Intel Xeon E5-2620 v4. A Nvidia GTX 1080 Ti with 11GB is used as the accelerator. Python2.7 is the main programming language for the experiment.

B. MULTIPLE-LAYER MODELS EVALUATION

The purpose of the experiment is to validate the performance of the proposed framework (DCF-IDS). In this experiment, we first compared the classification performance between the proposed DCF-IDS model with the GooLeNetNP only. NBC was used as a training and test dataset to classify 101 classes. Figure 7 shows the distribution of DR between DCF-IDS and GoogLeNetNP. Table 3 shows the comparison between DCF-IDS and GoogLeNetNP in DoS. Table 4 shows the average accuracy of both methods. The overall accuracy of both methods is listed in Affix Table 14.
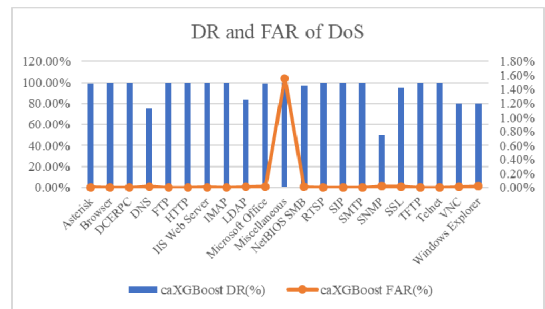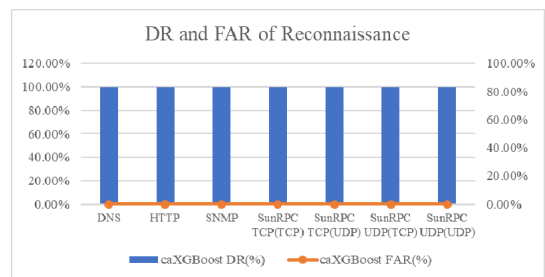

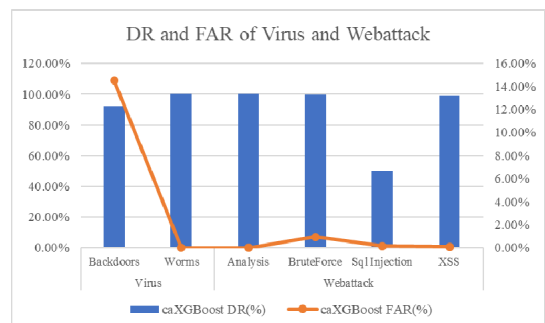(a) DR and FAR of DoS subclass detection
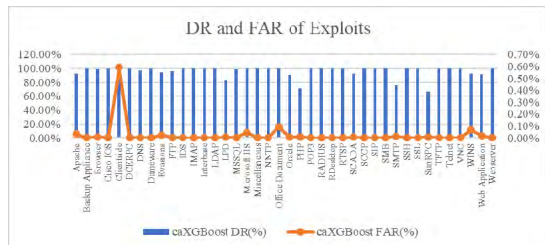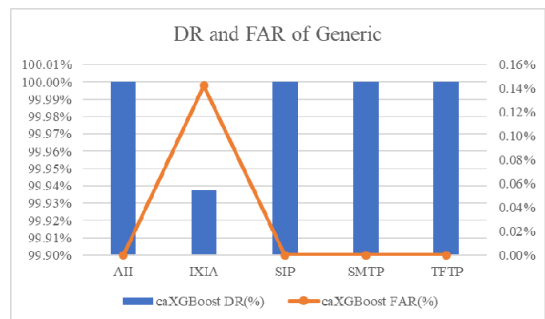
(b) DR and FAR of Exploits subclass detection

(c) DR and FAR of Generic subclass detection

(d) DR and FAR of Reconnaissance subclass detection

(e) DR and FAR of Virus and Webattack subclass detection

FIGURE 9. (a) DR and FAR of DoS subclass detection. (b) DR and FAR of Exploits subclass detection. (c) DR and FAR of Generic subclass detection. (d) DR and FAR of Reconnaissance subclass detection. (e) DR and FAR of Virus and Webattack subclass detection.

TABLE 7. DR of different CNN structures.

	AlexNet	VGG19	GoogLeNet	Inception V3	ResNet18	GoogLeNetNP
Normal	99.90%	99.26%	99.86%	99.90%	99.86%	99.92%
DoS	93.52%	10.87%	96.99%	57.34%	97.33%	98.01%
Exploits	98.17%	57.06%	98.48%	79.38%	98.76%	98.88%
Fuzzers	98.64%	19.42%	98.27%	97.04%	97.75%	99.13%
Generic	86.61%	11.26%	92.88%	82.75%	93.20%	94.73%
Reconnaissance	90.97%	0.00%	96.05%	64.74%	96.33%	95.20%
Shellcode	21.50%	3.74%	85.05%	57.94%	88.79%	86.92%
Virus	0.00%	0.00%	52.00%	6.00%	68.00%	66.00%
Webattack	98.33%	0.00%	99.12%	87.17%	99.21%	99.51%

TABLE 8. FAR of different CNN structures.

	AlexNet	VGG19	GoogLeNet	Inception V3	ResNet18	GoogLeNetNP
Normal	0.27%	5.01%	0.39%	0.35%	0.38%	0.22%
DoS	0.40%	5.92%	0.19%	2.66%	0.16%	0.12%
Exploits	0.33%	8.20%	0.28%	3.74%	0.23%	0.20%
Fuzzers	0.05%	3.52%	0.07%	0.12%	0.09%	0.03%
Generic	0.13%	1.06%	0.07%	0.18%	0.07%	0.05%
Reconnaissance	0.03%	0.39%	0.01%	0.12%	0.01%	0.02%
Shellcode	0.05%	0.07%	0.01%	0.03%	0.01%	0.01%
Virus	0.03%	0.04%	0.01%	0.03%	0.01%	0.01%
Webattack	0.01%	0.85%	0.01%	0.10%	0.01%	0.00%

TABLE 9. The comparison among three algorithms on Shellcode.

	caXGBoost	gcForest	GoogLeNetNP
Training Time (s)	4.42	11.95	360.00
Detecting Time (s)	0.25	4.11	5.27
Accuracy	95.79%	95.79%	43.4%

TABLE 10. The comparison among three algorithms on Fuzzers.

	caXGBoost	gcForest	GoogLeNetNP
Training Time (s)	40.83	120.25	3242.00
Detecting Time (s)	3.08	26.73	42.88
Accuracy	99.93%	95.03%	95.00%

From Table 4, we can see that the both average DR and FAR of DCF-IDS performs better than GoogLeNetNP. Traditional intrusion detection usually takes only one algorithms because of the few classes needed to be detected. When the number of classes increase, the model does not perform well. For GoogLeNetNP, the overall accuracy is 98.35%. According to Figure 7, there are 68 subclasses doesn't reach the 90% detection rate and among them, there are 21 subclasses with 0% detection rates. Thus, the combination of different representation learning algorithms will improve the feature extracting, the accuracy, the detection rate and the speed. For DCF-IDS, the overall accuracy is 99.24%. Only 22 subclasses do not achieve the 90% detection rate and the lowest detection rate is 61.90%, which has much better performance than the one of GoogLeNetNP.

C. DATA ENCODING EVALUATION

The purpose of the experiment is to evaluate the feature extraction performance of P-Zigzag on raw data traffic in deep neural networks and the cost time on converting, training and testing. Two comparative experiments have been conducted for evaluation of the effectiveness of the proposed method, in comparison with the existing encoding method - OHE (One-hot encoding (OHE plays the same role in preprocessing as P-Zigzag. It converts the raw package into a sparse matrix).

The first experiment computes the time cost of converting the raw traffic date (10360 DoS packets) into the matrix. The preprocessed datasets through two methods (P-Zigzag and OHE) were trained by the same classifier (GoogLeNetNP) respectively. The parameters take the default ones, that is, epoch is 50, learning rate is 0.001, optimizer takes SGD and batch size is 128. The training time and the testing time of the model were also recorded. The results of the experiment are shown in Table 4. The overall accuracy is listed in Table 5 and the detailed results are shown as below in Table 6.

TABLE 11. DR and FAR results of the comparison on Shellcode.

	caXGBoost		gcForest		GoogLeNet	
	DR(%)	FAR(%)	DR(%)	FAR(%)	DR(%)	FAR(%)
BSD	95.45%	0.89%	95.45%	0.88%	86.36%	7.69%
BSDi	100.00%	0.00%	93.75%	0.49%	0.00%	15.09%
Decoders	100.00%	0.00%	94.44%	0.56%	66.67%	8.00%
FreeBSD	100.00%	0.00%	87.50%	0.72%	0.00%	11.76%
Linux	100.00%	0.00%	100.00%	0.00%	10.53%	24.84%
Mac OS X	92.00%	1.16%	96.00%	0.58%	50.00%	15.56%
Multiple OS	90.00%	0.79%	90.00%	0.78%	0.00%	15.09%
NetBSD	87.50%	0.72%	87.50%	0.72%	0.00%	11.76%
OpenBSD	75.00%	0.94%	100.00%	0.00%	0.00%	8.16%
Solaries	83.33%	1.30%	91.67%	0.65%	0.00%	15.09%
Windows	100.00%	0.00%	100.00%	0.00%	86.67%	6.41%

TABLE 12. DR and FAR results of the comparison on Shellcode.

	caXGBoost		gcForest		GoogLeNet	
	DR(%)	FAR(%)	DR(%)	FAR(%)	DR(%)	FAR(%)
BGP	100.00%	0.00%	96.62%	1.40%	95.95%	1.66%
DCERPC	100.00%	0.00%	93.75%	0.10%	0.00%	1.51%
FTP	100.00%	0.00%	54.40%	1.74%	95.24%	0.19%
HTTP	100.00%	0.00%	96.82%	0.99%	100.00%	0.00%
OSPF	100.00%	0.00%	91.92%	0.22%	35.71%	1.62%
PPTP	100.00%	0.00%	96.34%	0.04%	96.97%	0.03%
RIP	98.70%	0.03%	97.40%	0.05%	100.00%	0.00%
SMB	100.00%	0.00%	96.87%	1.92%	98.29%	1.06%
Syslog	99.31%	0.01%	92.36%	0.12%	100.00%	0.00%
TFTP	100.00%	0.00%	100.00%	0.00%	100.00%	0.00%

According to the results, P-Zigzag performs much better than OHE in terms of the speed and precision, that is 3 times faster on converting, over 5 times faster on training, about 1.2 times faster on testing and the accuracy increased by more than 25%.

The overall accuracy of P-Zigzag is above 99%; in addition to the two categories of Shellcode and Virus, the accuracy of other categories is above 90%. The main reason for the lower accuracy of Shellcode and Virus is that data imbalance affects the detection of smaller categories of samples. The matrix size of OHE image is 256 × 1500, resulting in too much redundant information in the convolution process, so that will cost much process time and the memory of the machine and affect the feature extraction and learning. According to the characteristics of intrusion detection data, P-Zigzag algorithm concentrates data report information in a fixed matrix, which reduces information redundancy and improves the efficiency of feature extraction.

D. EVALUATION ON FIRST LAYER DETECTION MODEL

The purpose of this experiment is to compare the performance comparison between GoogLeNetNP and other commonly used network structures in convolutional neural networks to determine the model structure of the first-level attack large classifier in the cascade framework, including AlexNet [53], VGG19 [54], GoogLeNet, Inception V3 [55] and ResNet18, to determine the performance of the first layer classifier. Figure 8 shows the overall accuracy of different networks and Table 7 and Table 8 will show DR and FAR respectively. The parameters take the default ones, that is, epoch is 50, learning rate is 0.001, optimizer takes SGD and batch size is 128.

By contrast, GoogLeNetNP has a better accuracy than the ones of other classifiers. The performance of GoogLeNet,

TABLE 13. The detailed statistics of NBC.

Main Classes	Sub-classes	Resources	Training	Test
DoS	Asterisk	UNSW-NB15	306	40
	Browser	UNSW-NB15	284	50
	DCERPC	UNSW-NB15	14	5
	DNS	UNSW_NB15	20	6
	FTP	UNSW_NB15	56	15
	HTTP	UNSW_NB15	3362	332
	IIS Web Server	UNSW_NB15	1540	150
	IMAP	UNSW_NB15	8	2
	LDAP	UNSW_NB15	28	6
	Microsoft Office	UNSW_NB15	1074	206
	Miscellaneous	UNSW_NB15	28010	9974
	NetBIOS SMB	UNSW_NB15	144	30
	RTSP	UNSW_NB15	8	2
	SIP	UNSW_NB15	16	5
	SMTP	UNSW_NB15	30	8
	SNMP	UNSW_NB15	10	4
	SSL	UNSW_NB15	310	32
	Telnet	UNSW_NB15	8	2
	TFTP	UNSW_NB15	8	2
	VNC	UNSW_NB15	24	6
	Windows Explorer	UNSW_NB15	42	10
Exploits	Apache	UNSW_NB15	310	126
	Backup Appliance	UNSW_NB15	168	16
	Browser	UNSW_NB15	3071	320
	Cisco IOS	UNSW_NB15	162	26
	Clientside	UNSW_NB15	21930	12690
	Dameware	UNSW_NB15	10	4
	DCERPC	UNSW_NB15	270	20
	DNS	UNSW_NB15	64	16
	Evasions	UNSW_NB15	1148	128
	FTP	UNSW_NB15	198	26
	IDS	UNSW_NB15	24	8
	IMAP	UNSW_NB15	32	8
	Interbase	UNSW_NB15	140	16
	LDAP	UNSW_NB15	5450	452
	LDP	UNSW_NB15	52	12
	Microsoft IIS	UNSW_NB15	1064	114
	Miscellaneous	UNSW_NB15	9501	3064
	MSSQL	UNSW_NB15	12	4
	NNTP	UNSW_NB15	18	8
	Office Document	UNSW_NB15	25529	11267
	Oracle	UNSW_NB15	180	20
	PHP	UNSW_NB15	60	10
	POP3	UNSW_NB15	18	8
	RADIUS	UNSW_NB15	10	4
	RDesktop	UNSW_NB15	10	4
	RTSP	UNSW_NB15	36	8
	SCADA	UNSW_NB15	276	20
	SCCP	UNSW_NB15	14	4
	SIP	UNSW_NB15	36	10
	SMB	UNSW_NB15	1454	106
	SMTP	UNSW_NB15	136	16
	SSH	UNSW_NB15	30	8
	SSL	UNSW_NB15	18	8
	SunRPC	UNSW_NB15	27	8
	Telnet	UNSW_NB15	116	20
	TFTP	UNSW_NB15	24	6
	VNC	UNSW_NB15	20	6
	Web Appliance	UNSW_NB15	1200	252
	Webserver	UNSW_NB15	562	66
	WINS	UNSW_NB15	34	10
Fuzzers	BGP	UNSW_NB15	6739	1975
	DCERPC	UNSW_NB15	194	102
	FTP	UNSW_NB15	1504	252
	HTTP	UNSW_NB15	5668	1596
	OSPF	UNSW_NB15	794	168
	PPTP	UNSW_NB15	330	66
	RIP	UNSW_NB15	925	134
	SMB	UNSW_NB15	7534	2575

TABLE 13. *(Continued.)* The detailed statistics of NBC.

	Syslog	UNSW_NB15	578	106
	TFTP	UNSW_NB15	64	12
Generic	AII	UNSW_NB15	2194	474
	IXIA	UNSW_NB15	6410	1298
	SIP	UNSW_NB15	122	20
	SMTP	UNSW_NB15	648	64
	TFTP	UNSW_NB15	40	10
Reconnaissance	DNS	UNSW_NB15	12	4
	HTTP	UNSW_NB15	563	82
	SNMP	UNSW_NB15	24	8
	SunRPC TCP(TCP)	UNSW_NB15	508	186
	SunRPC TCP(UDP)	UNSW_NB15	496	192
	SunRPC UDP(TCP)	UNSW_NB15	622	68
	SunRPC UDP(UDP)	UNSW_NB15	630	62
Shellcode	BSD	UNSW_NB15	88	18
	BSDi	UNSW_NB15	32	8
	Decoders	UNSW_NB15	36	10
	FreeBSD	UNSW_NB15	16	6
	Linux	UNSW_NB15	78	16
	Mac OS X	UNSW_NB15	50	14
	Multiple OS	UNSW_NB15	20	8
	NetBSD	UNSW_NB15	16	6
	OpenBSD	UNSW_NB15	8	4
	Solaries	UNSW_NB15	24	8
	Windows	UNSW_NB15	60	16
Virus	Backdoors	UNSW_NB15	158	40
	Worms	UNSW_NB15	44	18
Webattack	Analysis	UNSW_NB15	106	16
	BruteForce	CIC-IDS2017	6504	1118
	Sql Injection	CIC-IDS2017	9	4
	XSS	CIC-IDS2017	1520	200
Normal	/	UNSW-NB15 CIC-IDS2017	246689	137248

ResNet18 and GoogLeNetNP are almost identical in detection accuracy, but in terms of the false alarm rate, GoogLeNetNP performs the best. The time cost of training among GoogLeNet, ResNet18 and GoogLeNetNP is 3h, 4h and 3h respectively. Considering the network model structure, GoogLeNetNP replaces large convolution with small convolution kernels, which can speed up the training and testing process. Thus, the first layer network model uses GoogLeNetNP as the classifier for the attack class.

E. EVALUATION ON SECOND LAYER DETECTION MODEL
The purpose of the experiment is to compare the detection performance of caXGBoost, gcForest and GoogLeNetNP on attack subclasses, including training time, detection time, overall accuracy, accuracy and FAR. By experimental comparison, the model classifiers of the second layer in the overall algorithm framework will be determined.

Table 9 and Table 10 show the comparison of the detection performance of caXGBoost, gcForest and GoogLeNetNP based on the Shellcode and Fuzzers attack. Shellcode has 11 subclasses, of which the number of training sets is 428 and the test set is 114. Fuzzers has 10 subclasses with 24330 training data and 6986 test data. The number of classifiers set for XGBoost is 11 and 10 respectively. The parameters take the default ones, that is, fold validation is 5, learning rate is 0.1, max layer is 5 and early stop is 3. The number of classifiers in each layer is determined by the number of subclasses.

It can be seen from the Table 9 that the test results of the two models caXGBoost and gcForest of the deep random forest network are better than the GoogLeNetNP of the deep learning network, which demonstrates that deep learning does not perform as well as deep random forest network on dealing with small datasets and data imbalances, and gives priority to deep random forest methods on such problems. Meanwhile, caXGBoost is superior to gcForest in terms of computing time for both training and testing or phases, and the detection time is increased by 16 times. In Table 10, when the amount of data increases, caXGBoost also performs better than gcForest, while the performance of deep learning network improves but still has poor detection in some subclasses because of the imbalanced data. The detailed results of DR and FAR is shown in Table 11 and Table 12.

gcForest uses multi-grained scanning to get more dimensions from the matrix, which increase the complexity of the model calculation and doesn't help improve the detection rate on 64×64 gray-scale image. The simple structure of caXGBoost is beneficial to the training and detection speed of the model, and the detection accuracy of the overall model has a good performance. It can be seen that the two-dimensional matrix image of network data conversion from table 11 and 12 is more suitable for the caXGBoost model network.

TABLE 14. The comparison between DCF-IDS and GoogLeNetNP.

Classes / Subclasses		DCF-IDS		GoogLeNetNP	
		DR(%)	FAR(%)	DR(%)	FAR(%)
Normal		99.88%	0.332%	99.93%	0.197%
DoS	Asterisk	98.69%	0.000%	97.37%	0.001%
	Browser	84.51%	0.004%	0.00%	0.027%
	DCERPC	100.00%	0.000%	0.00%	0.003%
	DNS	80.00%	0.001%	40.00%	0.002%
	FTP	92.86%	0.001%	35.71%	0.005%
	HTTP	99.58%	0.001%	98.21%	0.003%
	IIS Web Server	99.35%	0.001%	98.96%	0.001%
	IMAP	100.00%	0.000%	0.00%	0.001%
	LDAP	92.86%	0.000%	42.86%	0.002%
	Microsoft Office	94.97%	0.006%	70.15%	0.033%
	Miscellaneous	99.68%	0.018%	98.76%	0.070%
	NetBIOS SMB	86.11%	0.002%	25.00%	0.012%
	RTSP	100.00%	0.000%	0.00%	0.001%
	SIP	100.00%	0.000%	25.00%	0.002%
	SMTP	93.33%	0.000%	14.29%	0.004%
	SNMP	80.00%	0.000%	0.00%	0.002%
	SSL	98.71%	0.000%	100.00%	0.000%
	TFTP	75.00%	0.000%	0.00%	0.001%
	Telnet	100.00%	0.000%	100.00%	0.000%
	VNC	83.33%	0.001%	50.00%	0.002%
	Windows Explorer	61.90%	0.002%	0.00%	0.005%
Exploits	Apache	97.42%	0.002%	85.71%	0.010%
	Backup Appliance	96.43%	0.000%	73.81%	0.002%
	Browser	98.40%	0.003%	90.49%	0.016%
	Cisco IOS	97.53%	0.000%	27.50%	0.010%
	Clientside	94.66%	0.387%	97.37%	0.192%
	DCERPC	97.04%	0.000%	92.54%	0.000%
	DNS	100.00%	0.000%	75.00%	0.003%
	Dameware	100.00%	0.000%	50.00%	0.004%
	Evasions	97.91%	0.001%	83.97%	0.011%
	FTP	93.94%	0.001%	49.98%	0.007%
	IDS	100.00%	0.000%	50.00%	0.002%
	IMAP	87.50%	0.001%	50.00%	0.002%
	Interbase	100.00%	0.000%	88.57%	0.001%
	LDAP	99.89%	0.000%	99.41%	0.001%
	LPD	96.15%	0.000%	38%	0.004%
	MSSQL	100.00%	0.000%	0.00%	0.061%
	Microsoft IIS	99.25%	0.013%	92.86%	0.119%
	Miscellaneous	90.21%	0.000%	96.76%	0.000%

TABLE 14. *(Continued.)* The comparison between DCF-IDS and GoogLeNetNP.

	NNTP	100.00%	0.000%	0.00%	0.004%
	Office Document	96.62%	0.216%	98.59%	0.091%
	Oracle	96.67%	0.000%	40.00%	0.006%
	PHP	86.67%	0.001%	26.67%	0.004%
	POP3	100.00%	0.000%	0.00%	0.004%
	RADIUS	100.00%	0.000%	0.00%	0.002%
	RDesktop	80.00%	0.000%	0.00%	0.002%
	RTSP	88.89%	0.000%	66.67%	0.001%
	SCADA	95.65%	0.000%	52.17%	0.005%
	SCCP	71.43%	0.001%	66.67%	0.001%
	SIP	94.44%	0.000%	22.22%	0.004%
	SMB	98.35%	0.001%	91.18%	0.005%
	SMTP	89.71%	0.001%	35.29%	0.006%
	SSH	100.00%	0.000%	85.71%	0.001%
	SSL	100.00%	0.000%	0.00%	0.004%
	SunRPC	77.78%	0.001%	14.29%	0.004%
	TFTP	100.00%	0.000%	16.67%	0.009%
	Telnet	98.28%	0.000%	86.21%	0.000%
	VNC	90.00%	0.000%	0.00%	0.003%
	WINS	100.00%	0.000%	50.00%	0.068%
	Web Application	95.00%	0.002%	89.00%	0.004%
	Webserver	97.15%	0.000%	84.29%	0.001%
	BGP	99.73%	0.003%	97.80%	0.024%
	DCERPC	98.97%	0.001%	100.00%	0.000%
	FTP	99.07%	0.001%	94.68%	0.007%
	HTTP	99.86%	0.001%	99.79%	0.002%
	OSPF	100.00%	0.000%	98.99%	0.001%
Fuzzers	PPTP	100.00%	0.000%	95.12%	0.002%
	RIP	98.92%	0.001%	100.00%	0.000%
	SMB	99.89%	0.001%	99.15%	0.012%
	Syslog	99.31%	0.000%	100.00%	0.000%
	TFTP	96.88%	0.000%	100.00%	0.000%
	AII	100.00%	0.000%	100.00%	0.000%
	IXIA	98.35%	0.012%	94.82%	0.036%
Generic	SIP	98.36%	0.000%	100.00%	0.000%
	SMTP	93.52%	0.002%	55.56%	0.015%
	TFTP	90.00%	0.001%	70.00%	0.002%
	DNS	100.00%	0.000%	0.00%	0.002%
	HTTP	91.83%	0.004%	82.27%	0.008%
	SNMP	100.00%	0.000%	66.67%	0.001%
Reconnaissance	SunRPC TCP(TCP)	100.00%	0.000%	65.35%	0.035%
	SunRPC TCP(UDP)	100.00%	0.000%	70.16%	0.031%
	SunRPC UDP(TCP)	100.00%	0.000%	100.00%	0.000%
	SunRPC UDP(UDP)	100.00%	0.000%	100.00%	0.000%

TABLE 14. *(Continued.)* The comparison between DCF-IDS and GoogLeNetNP.

Shellcode	BSD	93.18%	0.001%	63.64%	0.004%
	BSDi	100.00%	0.000%	0.00%	0.004%
	Decoders	72.22%	0.001%	77.78%	0.001%
	FreeBSD	100.00%	0.000%	0.00%	0.003%
	Linux	94.87%	0.000%	63.16%	0.003%
	Mac OS X	88.00%	0.001%	50.00%	0.004%
	Multiple OS	90.00%	0.000%	0.00%	0.004%
	NetBSD	87.50%	0.000%	0.00%	0.003%
	OpenBSD	75.00%	0.001%	0.00%	0.002%
	Solaries	83.33%	0.001%	16.67%	0.004%
	Windows	100.00%	0.000%	100.00%	0.000%
Virus	Backdoors	82.28%	0.004%	79.49%	0.004%
	Worms	81.82%	0.002%	54.55%	0.004%
Webattack	Analysis	90.57%	0.001%	23.08%	0.007%
	BruteForce	99.88%	0.001%	98.83%	0.007%
	Sql Injection	77.78%	0.000%	0.00%	0.002%
	XSS	99.87%	0.000%	93.95%	0.007%

The subclass detection results of the rest classes are shown in Figure 9. The left vertical represents DR, the right vertical represents FAR, and the horizon represents subclass name. The line chart is the value of FAR and the histogram is the value of DR.

From the statistics of the experiments listed above, among 100 subclasses, there are 87 subclasses with detection rates exceeding 90% and the rest with detection rates from 50% to 90%. The worst detection results are Sql injection in Webattack and SNMP in DoS, whose detection rates are all 50% while the amount of training data is almost the least. Compared with the subclasses with the largest amount of training data in their corresponding attack classes (28010 in DoS and 6504 in Webattack), the extreme imbalanced data amount is one of the reasons leading to the poor detection. Besides, when the features of other subclasses are similar, the lack of training data will amplify the poor detection.

V. CONCLUSION
Due to the difficulty in feature extraction in the field of intrusion detection and the detection of fewer categories in past studies, we have proposed the DCF-IDS, which combines deep learning network with gcForest (deep random forest) to form a multiple-layer representation learning model. To the best of our knowledge, this is the first multiple-layer representation learning model for accurate end-to-end network intrusion detection by combining deep convolutional neural network with gcForest.. In addition, a novel encoding algorithm on converting the raw traffic data to gray-scaled images, P-Zigzag has been also proposed which is more effective and

faster than the existing encoding method (OHE). The proposed learning model uses improved GoogLeNetNP, as the first layer classifier to implement coarse-grained detection, which can identify N classes, including normal class. The second layer uses N-1 caXGBoost models, which runs parallelly for fine-grained detection to identify more subclasses. The method does not require any of the engineering techniques used in traditional intrusion detection methods. The experimental results show that the DCF-IDS significantly improves the accuracy and DR compared to the single algorithm or model. Furthermore, the DCF-IDS reduces the FAR because it automatically learns the features of raw traffic data, which improves the overall performance of the IDS.

In future, we will apply this proposed method to various datasets and networks for intrusion detection (e.g. IoT).

APPENDIX
See Tables 13 and 14.

REFERENCES
[1] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *Expert Syst. Appl.*, vol. 36, no. 10, pp. 11994–12000, 2009.
[2] G. Liu, Z. Yi, and S. Yang, "A hierarchical intrusion detection model based on the PCA neural networks," *Neurocomputing*, vol. 70, nos. 7–9, pp. 1561–1568, 2007.
[3] H. Nguyen, K. Franke, and S. Petrovic, "Improving effectiveness of intrusion detection by correlation feature selection," in *Proc. Int. Conf. Availability, Rel. Secur.*, Feb. 2010, pp. 17–24.
[4] A. H. Sung and S. Mukkamala, "Identifying important features for intrusion detection using support vector machines and neural networks," in *Proc. Symp. Appl. Internet*, Jan. 2003, pp. 209–216.
[5] J. Ryan, M.-J. Lin, and R. Miikkulainen, "Intrusion detection with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 1998, pp. 943–949.

[6] N. B. Amor, S. Benferhat, and Z. Elouedi, "Naive bayes vs decision trees in intrusion detection systems," in *Proc. ACM Symp. Appl. Comput.*, 2004, pp. 420–424.

[7] F. Zhang and D. Wang, "An effective feature selection approach for network intrusion detection," in *Proc. IEEE 8th Int. Conf. Netw., Archit. Storage*, Jul. 2013, pp. 307–311.

[8] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. W. M. van der Laak, B. van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Med. Image Anal.*, vol. 42, pp. 60–88, Dec. 2017.

[9] P. Chandna, M. Miron, J. Janer, and E. Gómez, "Monoaural audio source separation using deep convolutional neural networks," in *Proc. Int. Conf. Latent Variable Anal. Signal Separat.*, Feb. 2017, pp. 258–266.

[10] S. Rezaei and X. Liu, "Deep learning for encrypted traffic classification: An overview," *IEEE Commun. Mag.*, vol. 57, no. 5, pp. 76–81, May 2019.

[11] M. A. Salama, H. F. Eid, R. A. Ramadan, A. Darwish, and A. E. Hassanien, *Hybrid Intelligent Intrusion Detection Scheme*, vol. 96. Berlin, Germany: Springer, 2011, pp. 293–303.

[12] J. Kim, J. Kim, H. Le Thi Thu, and H. Lim, "Long short term memory recurrent neural network classifier for intrusion detection," in *Proc. Int. Conf. Platform Technol. Service (PlatCon)*, Feb. 2016, pp. 1–5.

[13] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," in *Proc. Int. Conf. Adv. Comput., Commun. Inform. (ICACCI)*, Sep. 2017, pp. 1222–1228.

[14] A. Y. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proc. 9th EAI Int. Conf. Bio-Inspired Inf. Commun. Technol.*, in Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, May 2016, pp. 21–26.

[15] W. Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," in *Proc. IEEE Symp. Secur. Privacy*, May 1999, pp. 120–132.

[16] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur, "Bayesian event classification for intrusion detection," in *Proc. 19th Annu. Comput. Secur. Appl. Conf.*, Dec. 2003, pp. 14–23.

[17] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in *Proc. Int. Joint Conf. Neural Netw.*, May 2002, pp. 1702–1707.

[18] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[19] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proc. 25th Int. Conf. Mach. Learn.*, Jul. 2008, pp. 160–167.

[20] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 1, pp. 41–50, Feb. 2018.

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

[22] S. Potluri and C. Diedrich, "Deep feature extraction for multi-class intrusion detection in industrial control systems," *Int. J. Comput. Theory Eng*, vol. 9, no. 5, pp. 374–379, Oct. 2017.

[23] S. Yadav and S. Subramanian, "Detection of application layer DDoS attack by feature learning using stacked autoencoder," in *Proc. Int. Conf. Comput. Techn. Inf. Commun. Technol. (ICCTICT)*, Mar. 2016, pp. 361–366.

[24] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21954–21961, 2017.

[25] X. Yuan, C. Li, and X. Li, "DeepDefense: Identifying DDoS attack via deep learning," in *Proc. IEEE Int. Conf. Smart Comput.*, May 2017, pp. 1–8.

[26] Z. Li, Z. Qin, K. Huang, and S. Ye, "Intrusion detection using convolutional neural networks for representation learning," in *Proc. Int. Conf. Neural Inf. Process.* Cham, Switzerland: Springer, 2017, pp. 858–866.

[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 630–645.

[29] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu, "HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2017.

[30] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman, "Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation," in *Proc. DARPA Inf. Survivability Conf. Expo.*, Jan. 2000, pp. 12–26.

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

[32] Z.-H. Zhou and J. Feng, "Deep forest," Feb. 2017, *arXiv:1702.08835*. [Online]. Available: https://arxiv.org/abs/1702.08835

[33] Z.-H. Zhou, *EEnsemble Methods: Foundations and Algorithms*. London, U.K.: Chapman & Hall, 2012.

[34] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[35] F. T. Liu, K. M. Ting, Y. Yu, and Z. H. Zhou, "Spectrum of variable-random trees," *J. Artif. Intell. Res.*, vol. 32, pp. 355–384, May 2008.

[36] L. Breiman, "Statistical modeling: The two cultures," *Stat. Sci.*, vol. 16, no. 3, pp. 199–231, 2001.

[37] K. Miller, C. Hettinger, J. Humpherys, T. Jarvis, and D. Kartchner, "Forward thinking: Building deep random forests," May 2017, *arXiv:1705.07366*. [Online]. Available: https://arxiv.org/abs/1705.07366

[38] J. Feng and Z.-H. Zhou, "Autoencoder by forest," in *Proc. 32nd AAAI Conf. Artif. Intell.*, Apr. 2018, pp. 2967–2973.

[39] J. Feng, Y. Yu, and Z.-H. Zhou, "Multi-layered gradient boosting decision trees," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 3551–3561.

[40] L. Han, Z. Haihong, Y. Erxin, B. Yuming, and L. Huiying, "A clothes classification method based on the gcForest," in *Proc. IEEE 3rd Int. Conf. Image, Vis. Comput. (ICIVC)*, Jun. 2018, pp. 429–432.

[41] W. Ma, H. Yang, Y. Wu, Y. Xiong, T. Hu, L. Jiao, and B. Hou, "Change detection based on multi-grained cascade forest and multi-scale fusion for SAR images," *Remote Sens.*, vol. 11, no. 2, p. 142, 2019.

[42] Q. Liu, H. Gao, Z. You, H. Song, and L. Zhang, "Gcforest-based fault diagnosis method for rolling bearing," in *Proc. Prognostics Syst. Health Manage. Conf. (PHM-Chongqing)*, Oct. 2018, pp. 572–577.

[43] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Trans. Comput.*, vol. C-23, no. 1, pp. 90–93, Jan. 1974.

[44] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[45] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, *arXiv:1312.4400*. [Online]. Available: https://arxiv.org/abs/1312.4400

[46] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 785–794.

[47] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proc. Mil. Commun. Inf. Syst. Conf. (MilCIS)*, Nov. 2015, pp. 1–6.

[48] N. Moustafa and J. Slay, "The evaluation of network anomaly detection systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set," *Inf. Syst. Secur.*, vol. 25, nos. 1–3, pp. 18–31, 2016.

[49] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th ICISSP*, Jan. 2018, pp. 108–116.

[50] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani, "Towards a reliable intrusion detection benchmark dataset," *Softw. Netw.*, vol. 2018, no. 1, pp. 177–200, 2018.

[51] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.

[52] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th Symp. Operating Syst. Des. Implement.*, 2016, pp. 265–283.

[53] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[54] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: https://arxiv.org/abs/1409.1556

[55] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2818–2826.

XUEQIN ZHANG received the Ph.D. degree in detection technology and automation devices from the East China University of Science and Technology (ECUST), Shanghai, China, in 2007. Since 1998, she has been with the Electrical and Communication Engineering Department, ECUST, where she is currently an Associate Professor. In 2006, she was a Visiting Scholar with the University of Wisconsin–Madison. Her research interests include information security, pattern classification, and datamining.

JIAHAO CHEN received the B.S. degree from the East China University of Science and Technology, in 2016, where he is currently pursuing the M.S. degree. He has been involved in numerous network security projects. His current research interests include intrusion detection, information security, and pattern recognition.

YUE ZHOU received the B.S. degree in microelectronics from the China Jiliang University, in 2017. He is currently pursuing the M.S. degree in communication and information engineering with the East China University of Science and Technology. He has been involved in several intrusion detection projects. His research interests include intrusion detection and machine learning.

LIANGXIU HAN received the Ph.D. degree from Fudan University, Shanghai, China, in 2002. She is currently with the School of Computing, Mathematics and Digital Technology, Manchester Metropolitan University. Her research interests include pattern classification, information security, and data mining. As a Principal Investigator (PI) or Co-PI, she has been conducting research in relation to big data processing, data mining, parallel and distributed computing/cloud computing (funded by EPSRC, BBSRC, Royal Society, Innovate UK, Horizon 2020, Industry, Charity, and Newton Fund).

JIAJUN LIN received the Ph.D. degree from Tsinghua University, Beijing, China, in 1998. He is currently a Professor with the School of Information Science and Engineering, East China University of Science and Technology, Shanghai, China. His research interests include neural networks and information security.

● ● ●