IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 28, 2021, accepted June 14, 2021, date of publication June 18, 2021, date of current version July 2, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3090464

Deep Learning and Regularization Algorithms for Malicious Code Classification

HAOJUN WANG[1], HAIXIA LONG[1], AILAN WANG[2], (Member, IEEE), TIANYUE LIU[1], AND HAIYAN FU[1]

[1]School of Information Science and Technology, Hainan Normal University, Haikou 571158, China
[2]Geneis Beijing Company Ltd., Beijing 100102, China

Corresponding author: Haixia Long (myresearch_hainnu@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61762034, in part by the Hainan Provincial Natural Science Foundation of China under Grant 618MS057 and Grant 620MS046, in part by the Hainan Provincial Innovation Research Team under Grant 2019CXTD405, in part by the Hainan Provincial Innovative Research Project for Postgraduates under Grant Hys2020-334, in part by the Hainan Provincial Reform in Education Project of China under Grant Hnjg2020-31 and Grant Hnjg2021ZD-15, and in part by the Education Department of Hainan Province of China under Grant Hnky2021-24.

ABSTRACT Network security has become a growing concern within the popularity and development of the Internet. Malicious code is one of the main threats to network security. Different types of malicious code have different functions and cause different harms. Therefore, improving the detection efficiency and recognition accuracy of malicious code is becoming an urgent problem to be solved. While traditional machine learning methods for malicious code detection largely depend on hand-designed features with experts' knowledge of the domain or focus on the images which come from malicious code binary files. These methods spend too much time on feature extraction. With the emergence of a large amount of malicious code data, the efficiency of traditional machine learning algorithms is getting worse and worse. In this paper, a workflow based on deep learning is proposed to detect and classify malicious codes. This workflow adopts a convolutional neural network (CNN) and the regularization algorithms to classify malicious code with N_gram semantic feature as input of the model. The convolutional neural network can automatically extract the features of malicious code while avoiding the need for manual feature selection. Regularization algorithms not only speed up the training process of the deep model but also improve the generalization ability in the case of effective prevention of over-fitting of the model. The proposed method is compared with the state-of-the-art methods and other deep learning models. Experimental results show that our workflow can improve the accuracy and efficiency of malicious code classification.

INDEX TERMS Malicious code classification, deep learning, convolutional neural networks, N-gram, regularization algorithm.

I. INTRODUCTION

Malicious Code has become one of the major threats to network security. It is a software or code fragment compiled to destroy software and hardware devices, stealing user information, disturbing user psychology, and interfering with normal use without authorization. Malicious code refers to the code that is artificially written or set and will cause harm to the network or system [1]. Broadly speaking, malicious code is also called malware which comes from merging the two words "Malicious" and "Software" [2] or malicious program. According to the definition of malicious code, malicious code includes but is not limited to Computer Virus, Worm, Trojan Horses, Rootkit, Spyware, Dishonest Adware, Crime ware, Logic Boom, Back Door, Botnet, Phishing, Malice Script, Spam, Malware in Intelligent Terminal Device and so on. Since malicious code is a program or code fragment with special functions, it often causes a lot of potential harms, such as destroying data, infringing on the system, stealing information, and leaking privacy. In addition, malicious code also has a unique ability to infect and spread. It can quickly infect hosts and spread quickly in the local area network or the internet.

With the emergence of malicious code technology, anti-malicious code technology has also emerged. The two technologies are mutually interdependent and evolve with each other. Malicious code detection is the primary technology against malicious code. Its main purpose is to analyze the

The associate editor coordinating the review of this manuscript and approving it for publication was Qi Zhou.

characteristic of software or code to determine whether it is malicious code. The accuracy of detection determines whether the harm caused by malicious code can be eliminated. Traditional malicious code detection technology requires manual feature extraction and identification of malicious code source files, and comparisons of the extracted features with a huge feature library to determine whether it is malicious code. Therefore, the detection efficiency of malicious code is relatively low and the accuracy of detection relies on manually extracted features.

To improve the accuracy and speed up the efficiency of malicious code detection, this paper presents malicious code detection technology based on deep learning models and regularization algorithms. Our research comes with the following contributions:

1) We provide a review of related work on malicious code classification.
2) We use operation codes that are generated during the disassembly of a large number of malicious code samples and then extracted operation code N-gram with different n values. Based on the operation code N-gram, convolutional neural network (CNN), Gated Recurrent Unit (GRU), and Support Vector Machine (SVM) are trained.
3) Our model does not require any domain expert knowledge, such as reverse engineering, binary disassembly, assembly language. The performance with accuracy 0.98 and Macro-F1-score 0.95 of our model exceeded some state-of-the-art methods.
4) Analyzing by performing classical machine learning and deep learning architectures on large datasets with a purpose to evaluate our proposed architectures in terms of their efficacy in dealing with large datasets of unknown types of malicious codes.

The rest of the paper is organized as follows. Section 2 presents related works of malicious code classification. In Section 3, we present the methods used in this paper which are consist of a convolutional neural network with regularization algorithms, features extraction, and evaluation metrics. Section 4 describes the experiments and results. Lastly, Section 5 summarizes our research, highlighting the limitation of the study and future research work.

II. RELATED WORK

Malicious code detection technology may be grouped into static and dynamic detection technology according to whether malicious code is running [3].

Static detection technology is a relatively basic and commonly used detection technology [4], [5]. In the case of not running malicious code, the structure, flow, and function of the unexecuted program are analyzed through techniques such as disassembly and decompilation to determine whether it is malicious code or contains malicious code fragments. Therefore, static detection is a well-established method for malicious code detection. Commonly used static detection technologies include signature detection technology [6], heuristic scanning technology [7], integrity detection technology, etc.

Dynamic detection technology [8]–[10] refers to whether the target program contains malicious behavior by observing and analyzing the behavior of the program and comparing the changes in the program's operating environment when running the target program. Determine whether there is malicious behavior by analyzing the characteristics of one or more executions of the target program. The dynamic detection technology can accurately detect the abnormal attributes of the program, but it cannot determine whether a particular attribute exists. Therefore, the dynamic detection technology is incomplete. Commonly used detection techniques include behavior monitoring detection, code simulation detection, etc.

In recent years, there have been many kinds of research on malicious code detection classification based on machine learning. Especially in the case of a large amount of data, many deep learning models [11] have highlighted their good ability to solve large-scale malicious code problems. Most of these studies can be grouped into static detection technology.

A. METHODS BASED ON CLASSIC MACHINE LEARNING

Naeem *et al.* [12] designed a malware classification system (MICS) that first extracted hybrid features of malware and then used SVM for classification. Their method achieved 97.4% classification accuracy while taking only 9339 samples from 25 malware families. While for small-scale analysis, their method achieved 99.6% classification accuracy by taking only 5116 samples from 10 malware families. Khalilian *et al.* [13] mined frequent sub-graphs from the operation code graphs of metamorphic malware and adopted J48, Naïve Bayes, and Logistic Regression to classify metamorphic malware. It is intended to alleviate the burden of human experts and underlying costs. Zhang *et al.* [14] proposed a static analysis method to classify ransomware families based on the N-gram of Operation code. They applied five machine learning algorithms (Decision Tree, Random Forest, K-Nearest neighbor, Naïve Bayes, and Gradient Boosting Decision Tree) to build a classifier with the best accuracy of 91.43%. But they cannot distinguish well all of the families of ransomware, like cryptowall, locky, and reveton. Liu *et al.* [15] proposed a multi-layer learning framework based on a bag-of-visual-words model to extract robust texture feature representations of malware gray scale images. This method has a high computational cost and may not be effective against malware disguised with obfuscation techniques.

B. METHODS BASED ON DEEP LEARNING

Yan *et al.* [16] proposed a deep neural network that took Convolutional Neural Networks (CNN) and Long-Short Term Memory (LSTM) networks to automatically learning features from the malicious files. It greatly reduced the cost of artificial features engineering. Gibert *et al.* [17] transformed the malware classification problem into time series

classification problem by representing malware executable as entropy value streams and then used CNN classifier to learn optimal discriminant subsequences of the time series. For the Microsoft dataset, this method achieved an accuracy of 98.28%. Nevertheless, it cannot work well for malware families with a small number of samples. For example, for the Simda family with only 42 samples, the classification accuracy was only 80.95%. Cui *et al.* [18] adopted the BAT Algorithm (BA) and combined with the Convolutional Neural Network (CNN) to improve the scheme proposed by Nataraj to solve the over-fitting problem caused by the uneven number of samples of malicious code family. Ni *et al.* [19] have designed a malware classification technique called MCSC (Malware Classification using SimHash and CNN), which converts the disassembled malware codes into gray images based on SimHash and then identifies their families by a convolutional neural network. Riaz Ullah Khan *et al.* [20] have utilized two distinct models which are GoogleNet and ResNet to identify the obscure or new sort of malware. They got a testing accuracy of 74.5% on GoogleNet and 88.36% precision on ResNet. Vinayakumar *et al.* [21] novelty in combining visualization and deep learning architectures for static, dynamic, and image processing-based hybrid approach applied in a big data environment is the first of its kind toward achieving robust intelligent zero-day malware detection. Overall, this paper paves way for effective visual detection of malware using a scalable and hybrid deep learning framework for real-time deployments. Liu *et al.* [22] proposed a visual detection method of malicious code based on adversarial training (AT) and CNN, which not only improved the detection accuracy of malware analysis but also prevented potential attacks of related variants. The method achieved up to 97.73% accuracy, along with 96.25% on average for all malware tested.

C. METHODS BASED ON IMPROVEMENT DEEP LEARNING

Sepideh Mohammadkhani and Esmaeilpour [23] proposed a new method for the detection of malware using reinforcement learning. This method combined the trial and error mechanism of reinforcement learning and the action optimization strategy, as well as the mining of in-depth features of the image by deep learning, to realize the identification of malicious code. Venkatraman *et al.* [24] presented the use of image-based techniques for detecting suspicious behavior of systems and proposed the application of hybrid image-based approaches with CNN-based deep learning architectures for effective malware classification. They proposed two CNN-based models: Unidirectional GRU (UniGRU) and Bidirectional GRU (BiGRU) models, also measured and compared the performance to other existing CNN architectures like Unidirectional LSTM (UniLSTM), and Bidirectional LSTM (BiLSTM). They performed experiments on two publicly available datasets: i) Microsoft Malware Classification Challenge (BIG, 2015) dataset and ii) Malimg dataset. Their model obtained ∼96% accuracy on average, however, the model did not consider the overhead time [24]. To ameliorate the lack of large publicly labeled datasets, Singh *et al.* proposed a GAN-based generative model for malware images and used the Malimg dataset to verify its effectiveness [25]. It can be used as a data augmentation technique to generate high-quality synthetic samples. Paper [26] investigates a central issue of how different hashing techniques can be combined to provide a quantitative malware score and to achieve better detection rates. They design and develop a novel approach for malware scoring based on the hashes results. The proposed approach is evaluated through several experiments. The evaluation demonstrates a significant improvement (>90%) in true detection rates of malware.

These technologies have broken through the shortcomings of traditional malicious code detection. In terms of analyzing a large number of variants and even unknown samples, and improving the speed and accuracy of detection, they have greatly improved the traditional malicious code detection technology. Traditional machine learning-based approaches often require pre-defining and extracting a set of features, which is computationally intensive and unsuitable for handling large amounts of data. Deep learning automates feature engineering, avoiding the high cost of manual feature selection and enabling the extraction of effective features.

III. METHODS

This paper presents a method to detect and classify malicious code using N_gram feature extraction and regularization convolutional neural network based on batch normalization. The structure diagram of the malicious code classification method in the present study is shown in Fig.1.

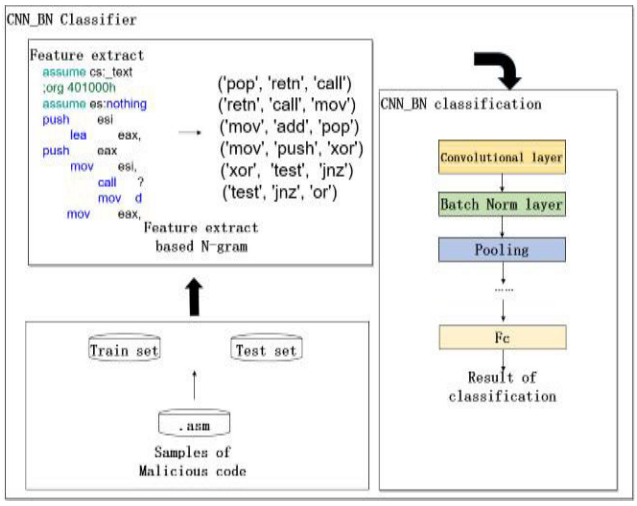

FIGURE 1. Structure of the malicious code classification method.

A. CONVOLUTIONAL NEURAL NETWORK (CNN)

The convolutional neural network inherits the common multilayer perceptual machine (MLP) structure as a feed-forward network. It contains multiple nonlinear feature transformations, where the parameters of the transformations are trained using the Gradient Descent (GD) method. In the feed-forward

neural network, the transformation process is as follows.

$$y = f\left(\sum_{i=1}^{M} w_i x_i + w_0\right) \tag{1}$$

where w_i are the parameters (weights) of the neural network, f is nonlinear functions that can approximate more complex functions as more layers of the model are added.

The convolutional neural network model mainly contains the following layers, as shown in Fig.2, usually convolutional, activation, and pooling layers together, and can contain multiple convolutional and fully connected layers.

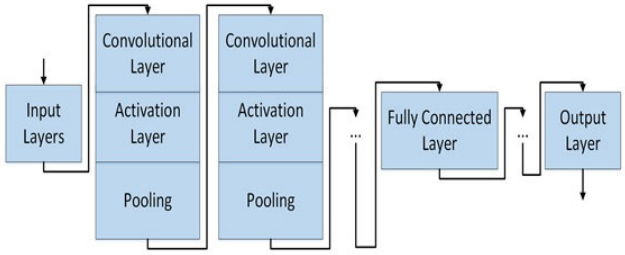

FIGURE 2. Convolutional neural network model.

The convolution layer uses a non-linear function and uses a convolution filter to move in a predefined window to extract features from data samples. The filter k discrete convolution performed the following transformation on the input I:

$$(I \times K)_{r,s} = \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K_{u,v} I_{r+u,s+v} \tag{2}$$

where h_1 and h_2 represent the size of convolution kernels, r, s is the position of input I.

The filter is derived from the following formula:

$$K = \begin{pmatrix} K_{-h_1,-h_2} & \cdots & k_{-h_1,h_2} \\ \vdots & K_{0,0} & \vdots \\ K_{h_1,-h_2} & \cdots & K_{h_1,h_2} \end{pmatrix} \tag{3}$$

The output of the convolutional layer (Y) consists of a series of feature diagrams, and the calculation formula for the *ith* feature diagrams is:

$$Y_i = B_i + \sum K_{i,j} \times X_j \tag{4}$$

where B is the bias value.

It is mainly characterized by local perception, weight sharing, and multi-convolution kernel. The former two mainly play a role in dimensionality reduction, while the latter provides specific operations for the re-extraction of features of different granularities. Fig.3. shows the structure of the convolutional layer. Through reasonable extraction of convolutional features, the convolutional neural network can help us distinguish the difference between malicious code and normal programs. The input matrix represents the features of the operation code extracted by N_gram, and the filter slides across the entire row of the matrix, similar to the application

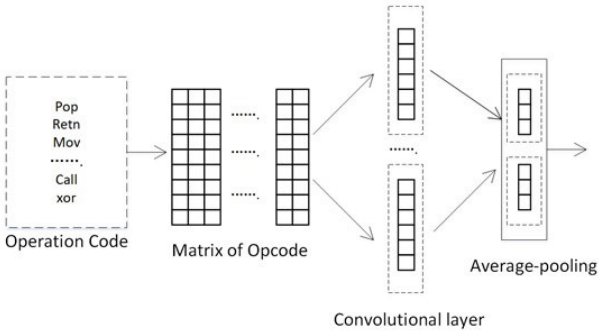

FIGURE 3. Convolutional layer structure.

in natural language processing (NLP). The width of the filter represents the width of the convolution kernel. Using multiple convolution kernels to perform convolution operations on the input sample matrix, and the size of the convolution kernel can be selected arbitrarily. In the experiment, we can compare the accuracy of malicious code detection and classification in the case of different filters by changing the size of the convolutional layer filter.

The pooling layer, also known as the down sampling layer, takes the feature results extracted from the convolution layer as input, and then further extracts the features to get deeper features. There are two main types: Maximum Pooling and Average Pooling. The former takes the maximum value in each sampling, and the latter takes the average value. The principle of the pooling layer can be expressed as:

$$pooling\,(x_{l-1})\,s_l = f\,(\beta_l \cdot pooling(x_{l-1}) + b_l) \tag{5}$$

where, $pooling(x_{l-1})$ is the pooling operation on the features of the $l-1$ layer, which is the output of the convolutional l layer, f is the activation function, β_l and b_l are used for bias of feature map output.

In general classification results were the output of the fully connected layer of the model. Because the classification of malicious codes is a multi-classification problem, and the features of different malicious codes are mutually exclusive, the Log softmax classifier is used for classification. It does one more log operation on the softmax result. The function can be expressed as the formula (6):

$$P(i) = \log \frac{\exp(\theta_i^T x)}{\sum_{k=1}^{K} \exp(\theta_k^T x)} \tag{6}$$

where, θ_i and x are column vectors. The output value of $P(i)$ has be scaled between 0 and 1 by the softmax function. In classification problems, θ is usually the parameter to be sought, and $P(i)$ is maximized by searching θ_i as the best parameter.

The loss function used in the model is the cross-entropy loss function, and its formula is as follows:

$$Loss = -\sum\nolimits_i q_i \lg \alpha_i \tag{7}$$

where, α_i is the confidence that the model predicts that it is the *ith* type of malicious code, and q_i indicates which type the malicious code sample belongs to by truth. If the sample is k, then $q_k = 1$ and other values are 0.

The activation function of the model uses Rectifier Linear Unit (*ReLU*). Compared with the activation functions such as Sigmoid and *Tanh*, the calculation of the *ReLU* is simpler, and the characteristics of the function help the training and convergence of the neural network model. The formula is shown as follows:

$$\mathrm{Re}\,LU(x) = \max(0, x) \tag{8}$$

B. BATCH NORMALIZATION

Each iteration of the neural network during the training process changes the parameters of each layer, which leads to a constant change in the distribution of the input data of the subsequent layers of the neural network, which will become more and more variable as the deep network is multi-layered, and to solve this problem, Batch Normalization is introduced, which has a regularization effect in the application. Normalization gives the data a zero mean and unit variance, and the data are transformed as follows.

$$x' = \frac{x - E(x)}{\sqrt{Var(x)}} \tag{9}$$

Assuming that the input data of each iteration during training are $\mathrm{X} = \{x_1, x_2, \ldots, x_m\}$, and the learning parameters are γ, β, the process of the Batch Normalization algorithm is as follows:

1) calculate the mean:

$$\mu_\beta = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{10}$$

2) calculate the variance:

$$\sigma_\beta^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\beta) \tag{11}$$

3) normalization:

$$\hat{x}_l = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \varepsilon}} \tag{12}$$

4) data scaling and translation:

$$y_i = \gamma \hat{x}_i + \beta \tag{13}$$

The Batch Normalization described above is the process during training, and only one sample at a time is predicted during the testing phase, so there is no μ_β and σ_β^2. At this point, the mean and variance computed during the training phase can be used. This is done by using the mean value calculated for each iteration of the training as the mean value for the test, and the variance is an unbiased estimate of the variance of each iteration.

Batch Normalization has the following advantages: 1) Gradient disappearance and gradient explosion can be avoided. The more and more skewed distribution is forced back to a more standard distribution so that the activated input value falls in the area where the nonlinear function is more sensitive to the input. In this way, a small change in the input will lead to a larger change in the loss function, which can make the gradient becomes larger and avoid the problem of gradient disappearance. 2) The training speed is accelerated. The larger the gradient means that the convergence rate of learning is fast, which can greatly accelerate the training speed. 3) Improving model generalization ability. Because batch standardization is not applied to the entire data set, but to mini-batch, some noise will be generated, which can improve the generalization ability of the model.

The loss function used in the model is cross-entropy. The cross-entropy is mainly used to determine the proximity of the actual output and the expected output, depicting the distance between the actual output probability and the expected output probability, i.e., the smaller the cross-entropy, the closer the two probability distributions, which is calculated by the following formula.

$$H(p, q) = -\sum_{x} (p(x)\log q(x) + (1 - p(x))\log(1 - q(x))) \tag{14}$$

In the formula, p represents the expected output, q is the actual output, and H (p, q) is the cross entropy.

C. CNN MALICIOUS CODE DETECTION CLASSIFICATION BASED ON BATCH NORMALIZATION

Since the executable program instructions of the malicious code are equivalent to a piece of text, when choosing a modeling method, there are certain similarities with the content of the processing text, and the convolution filter can be used to extract information and high-level feature detection for short text.

The N-gram-based convolutional neural network is composed of multiple convolutional layers and pooling layers. The malicious code classification model used in this implementation consists of two convolutional layer blocks, two pooling layers, and two full connection layers. Each convolution block is composed of two convolution layers, using 5×5 and 3×3 small convolution kernels respectively. Compared with the large convolution kernels, the small convolution kernels increase the depth of the neural network model and the number of nonlinear conversions, while ensuring that fewer parameters can be used to learn more complex features. The convolutional layer extracts features from the data while keeping the input size unchanged; the pooling layer plays the role in the second feature extraction; the fully connected layer is mainly to achieve the final classification. The detailed parameters are shown in Table 1.

The complete algorithm of CNN_BN is presented in Alg. 1

D. DATA SETS

The data set used in this article is from the malicious code classification data set used in the Kaggle Machine Learning

TABLE 1. Convolutional neural network model parameters.

Layers	Type	Kernels size	Padding/Stride
L1	Conv	5x5	2, 1
L2	Conv	5x5	2, 1
L3		Batch Normalization	
L4	Avgpooling	2x2	0, 1
L5	Conv	3x3	1, 1
L6	Conv	3x3	1, 1
L7		Batch Normalization	
L8	Avgpooling	2x2	0, 1
L9		Fully Connected	
L10		Fully Connected	

Algorithm 1 CNN_BN

Input: The malicious code file after the N-gram extraction feature, The number of iterations *epochs*
Output: Malicious code classification results *r*
1: **for** epoch in *epochs* **do**
2: **for** each malicious code x_i in malicious code **do**
3: The convolution layer convolves with x_i
4: Batch normalization is carried out for the result after convolution according to Eq.(10-13).
5: Further extraction of features according to Eq.(5).
6: Fully connected and Log softmax
7: get a result r_i
8: **end for**
9: Repeat step 2-7 until epoch = epochs, while disrupting the malicious code file
10: **end for**
11: get result *r*

Challenge, a machine learning-based data analysis competition hosted by Microsoft in 2015. This data set has been used in the papers of Gibert *et al.* [17], Kalash *et al.* [27], and Xiao *et al.* [28]. It contains 10868 malicious code samples and is divided into 9 different types. The size of the data is nearly 200 GB. In the current experiment, 80% of the 10868 samples are used as the training data set and the remaining 20% as the test set. Details are shown in Table 2.

The distribution of sample categories and quantity in the dataset is shown in Fig.4.

E. FEATURE EXTRACTION

In the entire process of malicious code detection and classification, feature extraction is one of the most important steps.

TABLE 2. Malicious code datasets.

serial number	Types of malicious code	Total number of samples	Number of training set samples	Number of test set samples
1	Ramnit	1541	1228	313
2	Lollipop	2478	1983	495
3	Kelihos ver3	2942	2336	606
4	Vendo	475	387	88
5	Simda	42	33	9
6	Tracur	751	595	156
7	Kelihos ver1	398	328	70
8	Obfuscator ACY	1228	995	233
9	Gatak	1013	809	204

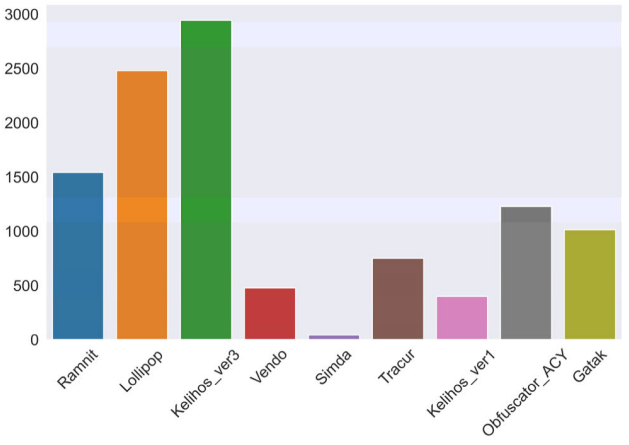

FIGURE 4. Dataset distribution.

The quality of feature selection directly affects the performance of model classification. This paper uses the Operation code feature extraction method based on N-gram.

Operation code is part of a machine language instruction. It selects the operation to be performed. A complete class of machine language instructions consists of a specification of one or more operands or an operation code. The operations of an operation code can include arithmetic, data manipulation, logical operations, and program control.

The N-gram model is widely used in natural language processing, information querying, bioengineering, and other fields. In [29], researchers presented technology that automatically detects the n-gram and clustering coefficient-based malware mutants and that automatically groups the different types of malware. They verified our system by applying more than 2600 malicious codes. The proposed technology does more than just respond to malware as it can also provide the ground for the effective analysis of new malware, the trend analysis of a malware group, the automatic identification of specific malware, and the analysis of the estimated trend of an attacker. In [30], a detailed investigation has been performed to evaluate the effectiveness of unigram, bigram, and trigram with stacked generalization. It's been found that with stacking, unigram provides more than 97% of accuracy which is the highest detection rate

against bigram and trigram. Wu *et al.* [31] present a malware classification method based on malware binaries, command sequences, and meta-features. They extracted key patterns of interaction behavior using an n-gram model. The results demonstrate 96.70% accuracy, with high precision and recall. Muhanmmad *et al.* [32] investigated an alternative method for malware detection that is based on N-grams and machine learning. They use a dynamic analysis technique to extract an Indicator of Compromise (IOC) for malicious files, which are represented using N-grams. In [33] researchers designed a situational awareness and analysis system for massive android malware detection which is using N-gram model extract features from App's smali code and DEX file. N-gram model is based on an assumption that words occurring at the *nth* position are only related to the (n − 1)th word, and the probability of occurrence of the whole sentence is the product of the probability of occurrence of each word. The main idea of n-gram is that, in a given text, starting from the first character of the text data, sliding on the text with the size of n characters, to produce a partially overlapping and continuous short segment of length N (gram). Using the N_gram model to express text information can improve the accuracy of text similarity measurement. Malicious code is essentially a text language, which also has structural and semantic features, so N_gram can be used as a feature analysis and extraction method for malicious code.

To extract operation code from PE files, we need to disassemble the samples. Disassembly translates the machine instructions stored in the PE files into a language that is more easily readable by human beings. In the experiment, we use IDA Pro to realize disassembly. Generate ASM file to store assembly instructions and extract operation code sequence from ASM file. Finally, the operation code N-gram is generated according to different n values. In the field of deep learning, a large number of features will not only increase the training time of the model, but also sometimes cannot improve the accuracy of the model, or even reduce the accuracy. Therefore, we select the operation code whose text frequency is more than 500 times is retained as a feature. The process is shown in Fig. 5.

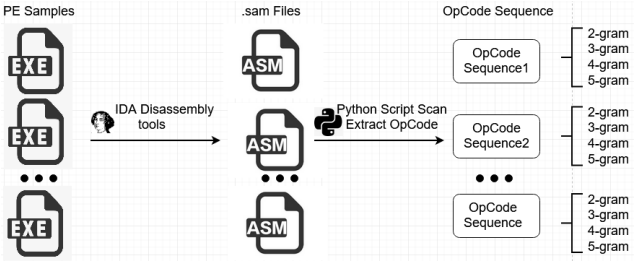

FIGURE 5. The process of the operation code generation.

In the experiment, we collected ASM files generated by disassembly from open sources and divided the training set and test set according to the ratio of 8:2. And the 2-gram, 3-gram, and 4-gram sequences of operation codes are extracted from each ASM file.

F. EVALUATION METRICS

The deep learning used in this article belongs to the field of machine learning, and malicious code detection belongs to the multi-classification problem in machine learning problems. We use the following evaluation metrics to evaluate the performance of the model.

TP: If the sample is a positive class and is predicted as a positive class, it is a true positive.

FP: If the sample is a negative class and is predicted as a positive class, it is called a false positive.

TN: If the sample is a negative class and is predicted to be a negative class, it is called a true negative class.

FN: If the sample is a positive class and is predicted as a negative class, it is called a false negative.

Accuracy: indicates the accuracy of classification, that is, in a given sample, the proportion of the number of correctly classified samples to the total sample. The calculation formula is:

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP} \tag{15}$$

Precision: refers to the proportion of truly correct samples among the samples whose prediction results are positive. The calculation formula is:

$$\Pr ecision = \frac{TP}{TP + FP} \tag{16}$$

Recall (recall rate): refers to the proportion of samples that are predicted to be positive in the total positive samples. The calculation formula is:

$$\mathrm{Re}\ call = \frac{TP}{TP + FN} \tag{17}$$

F1-score: It is a blend of precision and recall. Because the precision and recall are mutually exclusive, when one increases, the other one tends to decrease correspondingly. In order to reconcile the two indicators, F1-score is introduced, and its calculation formula is:

$$F1 = 2 \times \frac{\Pr ecision \times \mathrm{Re}\ call}{\Pr ecision + \mathrm{Re}\ call} \tag{18}$$

Macro-average F1: F1's macro average score is defined as the arithmetic average of the F1 scores of each category, that is, comprehensive and equal consideration of the accuracy of each category. It can well reflect the classification performance of the model for all classes, and its calculation formula is:

$$Macro - average = \frac{1}{n} \sum_{i=1}^{n} F_1 \tag{19}$$

ROC curve: refers to the receiver operating characteristic curve, which is a comprehensive indicator that reflects the sensitivity and specificity of continuous variables, and uses the composition method to reveal the relationship between sensitivity and specificity. The range of the horizontal and vertical coordinates of the ROC curve is [0, 1]. Generally, the larger the area formed by the ROC curve and the x-axis, the better the performance of the model.

IV. EXPERIMENTS AND RESULTS

To verify the performance of the sequences of operation codes which are extracted from ASM files, we use 2-gram, 3-gram, and 4-gram sequences of operation codes in experiments. Fig. 6 shows the accuracy on testing dataset of three N-grams on dataset. It can be observed that 3-gram is the most suitable sequences of operation codes.

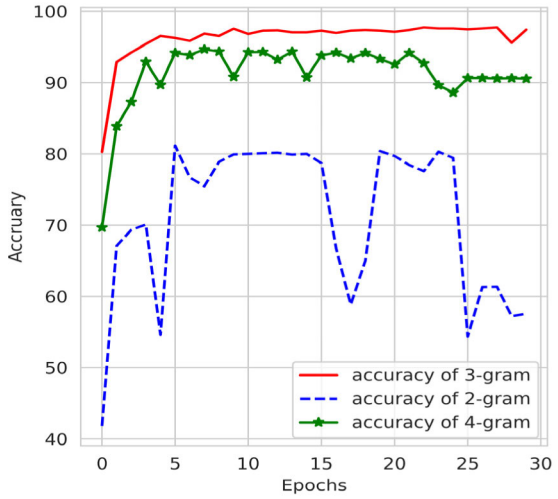

FIGURE 6. The accuracy curves of three N-gram.

To demonstrate the performance of the CNN_BN model, the experimental results of CNN_BN are compared with the results of the support vector machine of the traditional machine learning model, the GRU model based on the recurrent neural network model, and the convolutional neural network without Batch Normalization respectively.

Table 3 shows the performance of each machine learning model on the testing data set. The higher the value of Accuracy, Precision, Recall, and F1-score, the better the performance of the model is. It can be seen from Table 3 that the performance of the traditional machine learning model SVM is not as good as that of the deep learning model. In addition, the overall performance of the recurrent neural network is not as good as that of the convolutional neural network. Although CNN obtains the best precision value, we select the best model according to the Macro F1-score. This is because accuracy and precision can be misleading measures in datasets including class imbalance. For instance, a model can correctly predict the value of the majority class for all predictions and achieve high classification accuracy while making mistakes on the minority and critical classes. The macro F1-score metric penalizes this kind of behavior by calculating the metrics for each label and finding their un-weighted mean. The highest Macro F1-score (0.95) is reached by CNN_BN. So, the CNN_BN model has the best performance in malicious code detection.

To further evaluate the performance of our approach, we compared CNN_BN with state-of-the-art methods in the literature that have evaluated their models on the dataset

TABLE 3. Performance of the models.

model	Accuracy	Precision	Recall	Macro-F1-score
SVM	0.89	0.93	0.75	0.78
RNN_GRU	0.95	0.86	0.85	0.85
CNN	0.98	**0.97**	0.92	0.93
CNN_BN	**0.98**	0.96	**0.94**	**0.95**

TABLE 4. Performance of methods evaluated on the microsoft malware classification challenge.

model	Accuracy	Precision	Recall	Macro-F1-score
Drew[34]	0.97	-	-	-
Cui[18]	0.94	0.89	0.89	0.89
Venkatraman[21]	0.95	0.91	0.90	0.90
Ahmadi[35]	0.96	-	-	0.92
Le[36]	0.96	-	-	0.93
Gibert[37]	0.97	-	-	0.94
Yousefi-Azar[38]	0.93			0.86
CNN_BN	**0.98**	**0.96**	**0.94**	**0.95**

TABLE 5. Performance of CNN_BN on 9 categories.

Malicious code categories	Precision	Recall	F1-score
Ramnit	0.98	0.98	0.98
Lollipop	1.00	1.00	1.00
Kelihos ver3	1.00	1.00	1.00
Vendo	0.99	0.91	0.95
Simda	0.80	0.67	0.73
Tracur	0.85	1.00	0.92
Kelihos ver1	1.00	1.00	1.00
Obfuscator ACY	0.98	0.93	0.96
Gatak	1.00	0.99	1.00
Average	0.96	**0.94**	**0.95**

provided for Kaggle's Microsoft Malware Classification Challenge. The results are shown in Table 4. As a whole, CNN_BN algorithm obtains better results compared with state-of-the-art methods referring to the values of Accuracy (0.98) and Macro-F1-score (0.95). The values of Precision and Recall are not mentioning in the state-of-the-art methods.

Tables (5-8) show the classification performance of the models CNN_BN, CNN, GRU, and SVM on each category of malicious code, respectively. From the average values of nine categories, CNN_BN model achieves the best Recall (0.94) and F1-score (0.95), the CNN model obtains the best Precision (0.97). On the other hand, the Recall is 0.75 and the F1-score is 0.78 of the SVM algorithm, both of Recall and F1-scoce are 0.85 for GRU model. On the whole, the CNN_BN model is outperforming the other three models. Neural networks and in particular convolutional neural networks have recently attracted the academic community due to their advantages in processing raw data and their ability to learn features by themselves.

To show the performance of the three neural networks in malicious code classification more intuitively, Fig.7 shows the loss curves and accuracy curves of the three neural network models on the training data sets and the testing data sets. Fig.7 (a) and (b) show the loss curves and accuracy curves of the three kinds of neural networks on the training data

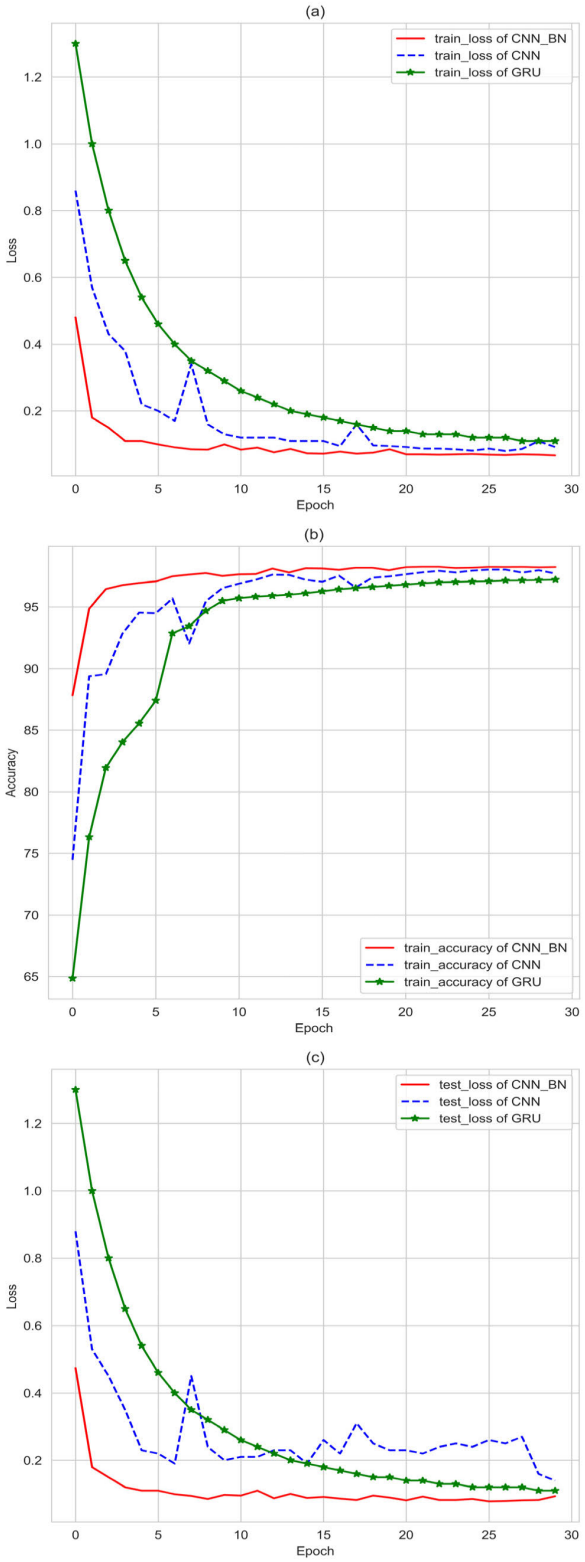

FIGURE 7. The loss curve and accuracy curve of the three neural network models. (a) and (b) on the training data sets, (c) and (d) on the test data sets.

sets, and Fig.7 (c) and (d) show the loss curves and accuracy curves on the testing data sets respectively. The smaller the loss value is, or the greater the accuracy value is, the better

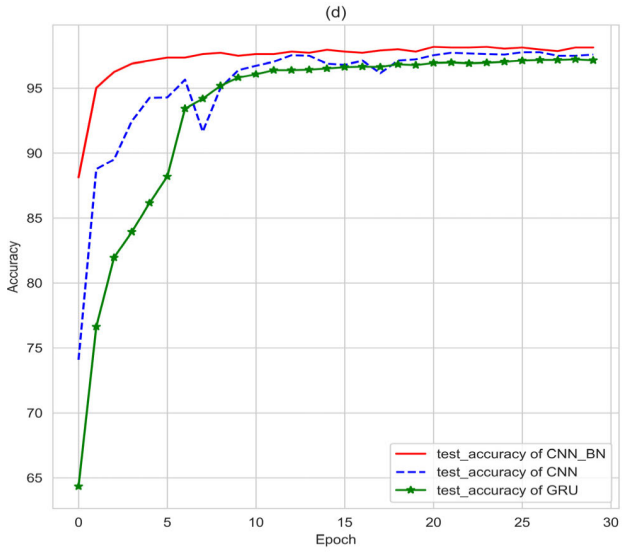

FIGURE 7. *(Continued.)* The loss curve and accuracy curve of the three neural network models. (a) and (b) on the training data sets, (c) and (d) on the test data sets.

TABLE 6. Performance of CNN on 9 categories.

Malicious code categories	Precision	Recall	F1-score
Ramnit	0.97	0.96	0.97
Lollipop	0.99	1.00	0.99
Kelihos ver3	1.00	1.00	1.00
Vendo	0.99	0.90	0.94
Simda	1.00	0.50	0.67
Tracur	0.83	0.99	0.90
Kelihos ver1	1.00	0.99	0.99
Obfuscator ACY	0.98	0.92	0.95
Gatak	0.98	0.99	0.99
Average	**0.97**	0.92	0.93

TABLE 7. Performance of GRU on 9 categories.

Malicious code categories	Precision	Recall	F1-score
Ramnit	0.97	0.99	0.98
Lollipop	1.00	1.00	1.00
Kelihos ver3	1.00	1.00	1.00
Vendo	0.97	0.81	0.89
Simda	0.00	0.00	0.00
Tracur	0.77	0.99	0.86
Kelihos ver1	1.00	0.97	0.99
Obfuscator ACY	1.00	0.91	0.95
Gatak	1.00	0.99	0.99
Average	0.86	0.85	0.85

the classification performance of the model will be. As shown in Fig.7 (a) and (c), the red curve is remarkably smaller than the blue or green line. It can be seen from Fig.7 (b) and (d), the red line is greater than the blue or green line. The results once again indicating that the proposed predictor is indeed much better than GRU and CNN predictors. Most important of all, the error of the convolutional neural network is indeed reduced by Batch Normalization.

To provide an intuitive comparison, the graph of Receiver Operating Characteristic (ROC) is utilized to show the advantage of CNN_BN. Fig.8 shows the ROC curves for the four

TABLE 8. Performance of SVM on 9 categories.

Malicious code categories	Precision	Recall	F1-score
Ramnit	0.97	0.96	0.97
Lollipop	1.00	0.98	0.99
Kelihos ver3	1.00	1.00	1.00
Vendo	1.00	0.48	0.65
Simda	1.00	0.64	0.78
Tracur	0.94	0.11	0.20
Kelihos ver1	1.00	0.65	0.79
Obfuscator ACY	0.49	0.98	0.65
Gatak	0.99	0.98	0.99
Average	0.93	0.75	0.78

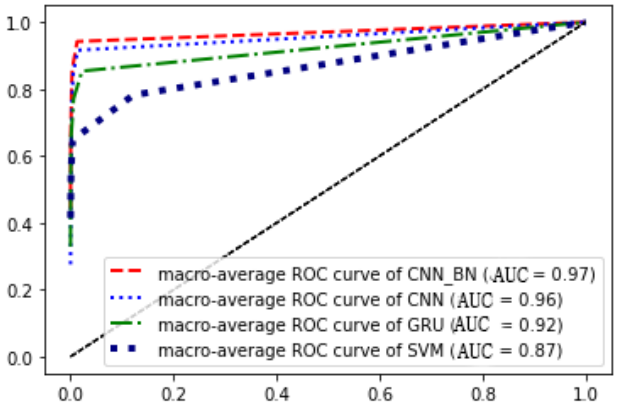

FIGURE 8. ROC curves for the four models.

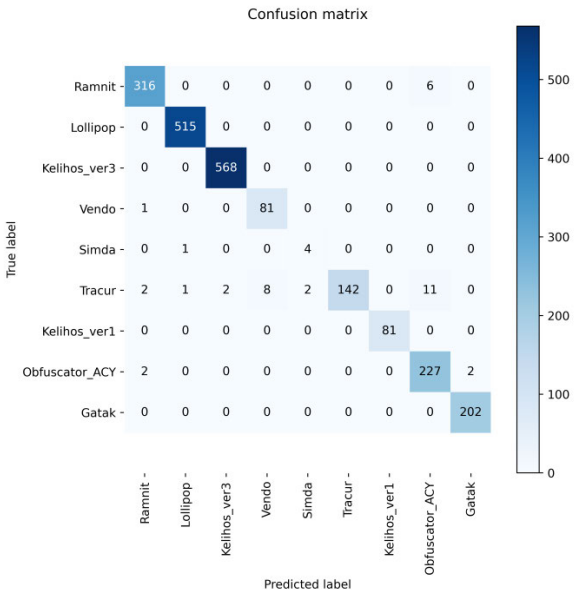

FIGURE 9. A confusion matrix for CNN_BN.

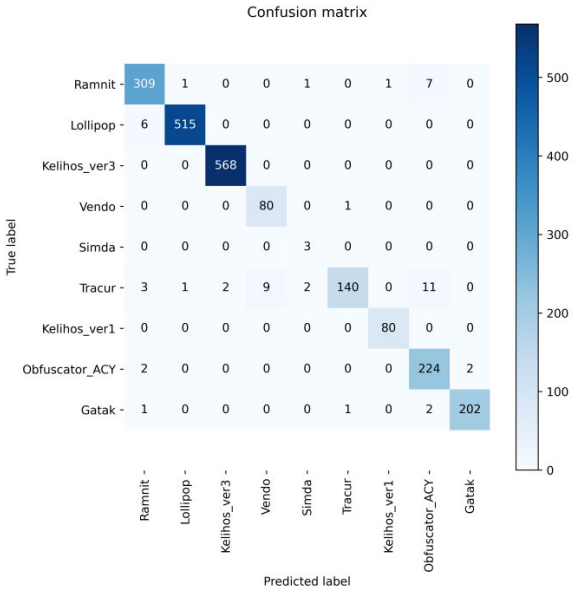

FIGURE 10. A confusion matrix for CNN.

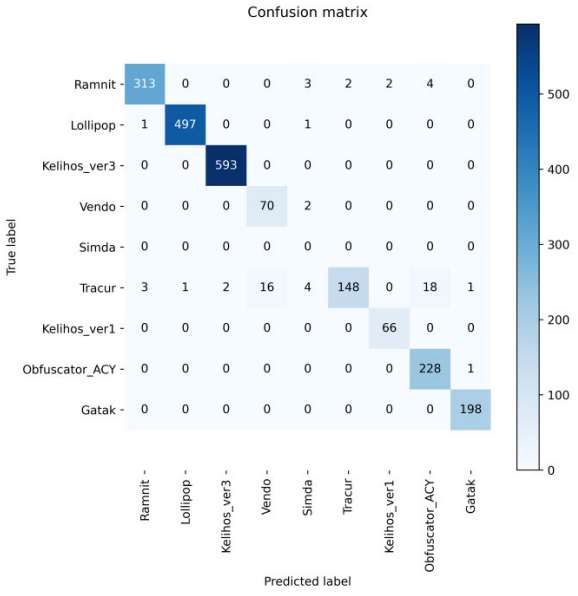

FIGURE 11. A confusion matrix for GRU.

models. The area under the ROC curve is called AUC (area under the curve). The greater the AUC formed by the ROC curve with the abscissa, the better the predictor will be. As we can be seen, the AUC value of CNN_BN is 0.97 which is remarkably greater than SVM and GRU. Compared with the AUC value (0.96) of CNN model, although CNN_BN has a little improvement, it is anticipated that CNN_BN will become a useful high throughput tool in this important area, or at the very least, play a complementary role to the existing methods.

The confusion matrix values are composed of the true position rate and the false negative rate of the malicious code classification. The abscissa in the confusion matrix represents the neural network prediction classification, the ordinate represents the true classification, and the numbers on the diagonal mean the number of correct classifications by the neural network. The numbers outside the diagonal represent the number of inconsistencies between the predicted classifications and the true classifications, indicating the number of incorrect classifications by the neural network. Figs. 9-12 show the confusion matrix of the four algorithms in the

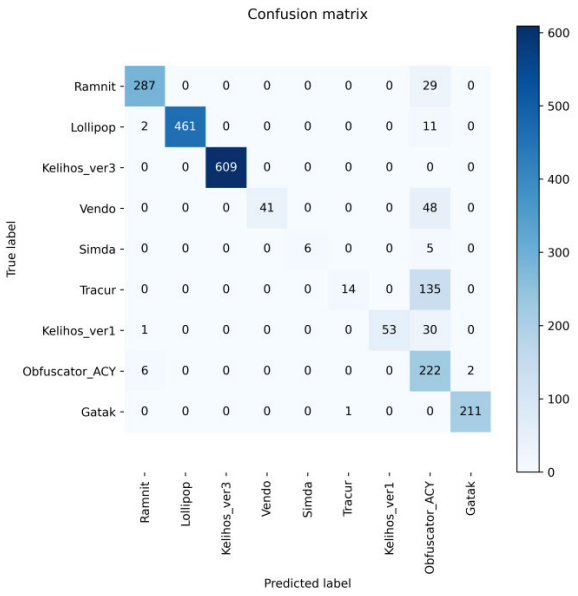

FIGURE 12. A confusion matrix for SVM.

9 types of malicious code detection and classification results on test data sets. The total sample size is 2174. The number of misclassification of samples is respectively 38, 53, 71, and 269 for CNN_BN, CNN, GRU, and SVM. Fig. 9 shows the confusion matrix of the CNN_BN algorithm. The major source of errors comes from the misclassification of samples belonging to the Tracur family, which 26 out of the 168 Tracur samples have been incorrectly classified. From the results of the confusion matrix, it can be concluded that the CNN_BN model has the best performance on test data sets.

V. CONCLUSION AND DISCUSSIONS

This paper presents a classification method for malicious code detection based on Operation code N-gram semantic feature extraction method combined with deep convolutional neural network and Batch Normalization regularization algorithm (CNN_BN). The method used in this paper achieves a classification accuracy of 0.98 and achieves macro-F1-score is 0.95 on Microsoft's public dataset compared with state-of-the-art methods [18]–[21], [34]–[38] in Table 4. Furthermore, to verify the performance of the CNN_BN model, the experimental results of CNN_BN are also compared with the results of the support vector machine of the traditional machine learning model, the GRU based on the recurrent neural network model, and the convolutional neural network without Batch Normalization respectively. Tables 5, 6, 7, and 8 show most of the values Precision, Recall, F1-score of CNN_BN has reached 1.00 on compared with other deep models (CNN and GRU) and traditional machine learning (SVM).

The method used in this paper is effective in malicious code detection. Because, on the one hand, convolutional neural networks have recently attracted the academic community due to their advantages on processing raw data and their ability to learn features by themselves, on the other, regularization can accelerate the convergence rate of the training process and prevent over fitting of the deep model.

The trained CNN_BN model has good generalization ability. For unknown types of malicious codes, according to our approach, at first, the features of a malicious code file can be extracted by the N-gram method. Then, the extracted malicious code features are classified by CNN_BN which is trained by datasets in the article. Finally, fully connected layers of deep learning models have the ability to predict.

Taken together, we can also draw the following further research directions: Firstly, in the N-gram feature extraction method used in this article, N is set to 3 and the frequency statistical threshold between operation code is 500. We can try other numerical combinations, and the frequency threshold is set as a variable based on the size of the malicious code file, rather than a constant value. Secondly, this article uses in-set verification instead of out-of-set verification, so there will be a problem of a relatively single data set, and different data sets can be used to further verify the methods applied in this article. Thirdly, the data set used in this experiment has the problem of data imbalance, but this problem has not been dealt with. Although the result was not bad, this problem also needed attention and resolution.

ACKNOWLEDGMENT

The authors appreciate Jialiang Yang and others for useful discussions.

REFERENCES

[1] G. Wang, T. Lu, and H. Yin, "Detection technology of malicious code family based on BiLSTM-CNN," *J. Phys., Conf. Ser.*, vol. 1650, Oct. 2020, Art. no. 032078.

[2] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture," *Comput. Netw.*, vol. 171, Apr. 2020, Art. no. 107138.

[3] S. Venkatraman and M. Alazab, "Use of data visualisation for zero-day malware detection," *Secur. Commun. Netw.*, vol. 2018, pp. 1–13, Dec. 2018.

[4] M. Amin, T. A. Tanveer, M. Tehseen, M. Khan, F. A. Khan, and S. Anwar, "Static malware detection and attribution in Android byte-code through an end-to-end deep system," *Future Gener. Comput. Syst.*, vol. 102, pp. 112–126, Jan. 2020.

[5] B. C. Yu, P. S. Song, and X. Xu, "An Android malware static detection scheme based on cloud security structure," *Int. J. Secur. Netw.*, vol. 13, no. 1, pp. 51–57, Jan. 2018.

[6] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *J. Comput. Virol. Hacking Techn.*, vol. 13, no. 1, pp. 1–12, Feb. 2017.

[7] B. Zhang, Q. Li, and Y. Ma, "Research on dynamic heuristic scanning technique and the application of the malicious code detection model," *Inf. Process. Lett.*, vol. 117, pp. 19–24, Jan. 2017.

[8] S. S. Alotaibi, "Regression coefficients as triad scale for malware detection," *Comput. Electr. Eng.*, vol. 90, Mar. 2021, Art. no. 106886.

[9] S. M. Bidoki, S. Jalili, and A. Tajoddin, "PbMMD: A novel policy based multi-process malware detection," *Eng. Appl. Artif. Intell.*, vol. 60, pp. 57–70, Apr. 2017.

[10] Z. Salehi, A. Sami, and M. Ghiasi, "MAAR: Robust features to detect malicious activity based on API calls, their arguments and return values," *Eng. Appl. Artif. Intell.*, vol. 59, pp. 93–102, Mar. 2017.

[11] M. Kalash, M. Rochan, N. Mohammed, N. Bruce, Y. Wang, and F. Iqbal, "A deep learning framework for malware classification," *Int. J. Digit. Crime Forensics*, vol. 12, no. 1, pp. 90–108, Jan. 2020.

[12] H. Naeem, B. Guo, and M. R. Naeem, "A light-weight malware static visual analysis for IoT infrastructure," in *Proc. Int. Conf. Artif. Intell. Big Data (ICAIBD)*, May 2018, pp. 240–244.
[13] A. Khalilian, A. Nourazar, M. Vahidi-Asl, and H. Haghighi, "G3MD: Mining frequent opcode sub-graphs for metamorphic malware detection of existing families," *Expert Syst. Appl.*, vol. 112, pp. 15–33, Dec. 2018.
[14] H. Zhang, X. Xiao, F. Mercaldo, S. Ni, F. Martinelli, and A. K. Sangaiah, "Classification of ransomware families with machine learning based on N-Gram of opcodes," *Future Gener. Comput. Syst.*, vol. 90, pp. 211–221, Jan. 2019.
[15] Y.-S. Liu, Y.-K. Lai, Z.-H. Wang, and H.-B. Yan, "A new learning approach to malware classification using discriminative feature extraction," *IEEE Access*, vol. 7, pp. 13015–13023, 2019.
[16] J. Yan, Y. Qi, and Q. Rao, "Detecting malware with an ensemble method based on deep neural network," *Secur. Commun. Netw.*, vol. 2018, pp. 1–16, Mar. 2018.
[17] D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Classification of malware by using structural entropy on convolutional neural networks," in *Proc. 32nd AAAI Conf. Artif. Intell., (AAAI), 30th Innov. Appl. Artif. Intell. (IAAI), 8th AAAI Symp. Educ. Adv. Artif. Intell. (EAAI)*, New Orleans, LA, USA, 2018, pp. 7759–7764.
[18] Z. Cui, F. Xue, X. Cai, Y. Cao, G.-G. Wang, and J. Chen, "Detection of malicious code variants based on deep learning," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3187–3196, Jul. 2018.
[19] S. Ni, Q. Qian, and R. Zhang, "Malware identification using visualization images and deep learning," *Comput. Secur.*, vol. 77, pp. 871–885, Aug. 2018.
[20] R. U. Khan, X. Zhang, and R. Kumar, "Analysis of ResNet and GoogleNet models for malware detection," *J. Comput. Virol. Hacking Techn.*, vol. 15, no. 1, pp. 29–37, Mar. 2019.
[21] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46717–46738, 2019.
[22] X. Liu, Y. Lin, H. Li, and J. Zhang, "A novel method for malware detection on ML-based visualization technique," *Comput. Secur.*, vol. 89, Feb. 2020, Art. no. 101682.
[23] S. Mohammadkhani and M. Esmaeilpour, "A new method for behavioural-based malware detection using reinforcement learning," *Int. J. Data Min. Model. Manag.*, vol. 10, no. 4, pp. 314–330, 2018.
[24] S. Venkatraman, M. Alazab, and R. Vinayakumar, "A hybrid deep learning image-based analysis for effective malware detection," *J. Inf. Secur. Appl.*, vol. 47, pp. 377–389, Aug. 2019.
[25] A. Singh, D. Dutta, and A. Saha, "MIGAN: Malware image synthesis using GANs," in *Proc. 33rd AAAI Conf. Artif. Intell., AAAI, 31st Innov. Appl. Artif. Intell. Conf., IAAI, 9th AAAI Symp. Educ. Adv. Artif. Intell., (EAAI)*, Honolulu, HI, USA, 2019, pp. 10033–10034.
[26] A. P. Namanya, I. U. Awan, J. P. Disso, and M. Younas, "Similarity hash based scoring of portable executable files for efficient malware detection in IoT," *Future Gener. Comput. Syst.*, vol. 110, pp. 824–832, Sep. 2020.
[27] M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang, and F. Iqbal, "Malware classification with deep convolutional neural networks," in *Proc. 9th IFIP Int. Conf. New Technol., Mobility Secur. (NTMS)*, Feb. 2018, pp. 1–5.
[28] G. Xiao, J. Li, Y. Chen, and K. Li, "MalFCS: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks," *J. Parallel Distrib. Comput.*, vol. 141, pp. 49–58, Jul. 2020.
[29] T. Lee, B. Choi, Y. Shin, and J. Kwak, "Automatic malware mutant detection and group classification based on the N-Gram and clustering coefficient," *J. Supercomput.*, vol. 74, no. 8, pp. 3489–3503, Aug. 2018.
[30] T. Islam, S. S. M. M. Rahman, M. A. Hasan, A. S. M. M. Rahaman, and M. I. Jabiullah, "Evaluation of N-Gram based multi-layer approach to detect malware in Android," *Procedia Comput. Sci.*, vol. 171, pp. 1074–1082, Jan. 2020.
[31] C.-J. Wu, S.-Y. Huang, K. Yoshioka, and T. Matsumoto, "IoT malware analysis and new pattern discovery through sequence analysis using meta-feature information," *IEICE Trans. Commun.*, vol. E103.B, no. 1, pp. 32–42, 2020.
[32] M. Ali, S. Shiaeles, G. Bendiab, and B. Ghita, "MALGRA: Machine learning and N-Gram malware feature extraction and detection system," *Electronics*, vol. 9, no. 11, p. 1777, Oct. 2020.
[33] Y. Zhang, W. Ren, T. Zhu, and Y. Ren, "SaaS: A situational awareness and analysis system for massive Android malware detection," *Future Gener. Comput. Syst.*, vol. 95, pp. 548–559, Jun. 2019.
[34] J. Drew, M. Hahsler, and T. Moore, "Polymorphic malware detection using sequence classification methods and ensembles," *EURASIP J. Inf. Secur.*, vol. 2017, no. 1, pp. 1–12, Dec. 2017.
[35] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *Proc. 6th ACM Conf. Data Appl. Secur. Privacy*, Mar. 2016, pp. 183–194.
[36] Q. Le, O. Boydell, B. Mac Namee, and M. Scanlon, "Deep learning at the shallow end: Malware classification for non-domain experts," *Digit. Invest.*, vol. 26, pp. S118–S126, Jul. 2018.
[37] D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Using convolutional neural networks for classification of malware represented as images," *J. Comput. Virol. Hacking Techn.*, vol. 15, no. 1, pp. 15–28, Mar. 2019.
[38] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, "Autoencoder-based feature learning for cyber security applications," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 3854–3861.

HAOJUN WANG was born in Zhejiang, China, in 1997. He received the B.S. degree in information management and information system from Yuncheng University. He is currently pursuing the master's degree with the School of Information Science and Technology, Hainan Normal University. His research interests include malware analysis, deep learning, artificial intelligence, and cyber space security.

HAIXIA LONG was born in Jiangsu, China, in 1980. She received the Ph.D. degree in light industry information technology and engineering from Jiangnan University, in 2010. She is currently a Professor with the School of Information Science and Technology, Hainan Normal University. Her research interests include deep learning, artificial intelligence, and cyber space security.

AILAN WANG (Member, IEEE) received the Ph.D. degree in bioinformatics from Northwest A&F University. She is currently the Senior Research Manager of Geneis Beijing Company Ltd., China. Her research interests include bioinformatics, biostatics, machine learning, and deep learning.

TIANYUE LIU was born in Chongqing, China, in 1996. She received the B.S. degree in information security from the Chongqing University of Posts and Telecommunications. She is currently pursuing the master's degree with the School of Information Science and Technology, Hainan Normal University. Her research interests include flow analysis, artificial intelligence, and cyber space security.

HAIYAN FU was born in Shandong, China, in 1978. She received the Ph.D. degree in system theory from Shandong University, in 2009. She is currently a Professor with the School of Information Science and Technology, Hainan Normal University. Her research interests include artificial intelligence and data mining.