Journal of Network Intelligence
Taiwan Ubiquitous Information
 ISSN 2414-8105
Volume **10**, Number **2**, May **2025**

Optimized Deep Learning-based Software Security Detection in Data-Driven Mode

Wei-Wei Ma*
School of Computer and Communication
Jiangsu Vocational College of Electronics and Information, Huai'an 223003, P. R. China
marqul2003@163.com

Yan Gao
School of Architecture, Decoration and Art Design
Jiangsu Vocational College of Electronics and Information, Huai'an 223003, P. R. China
84013944@qq.com

Da Huang
Sehan University, Jeollanam-do 58447, South Korea
57408943@qq.com

*Corresponding author: Wei-Wei Ma

Received February 20, 2024, revised June 31, 2024, accepted November 02, 2024.

Abstract. Artificial intelligence technology plays an important role in detecting and preventing malware. Deep learning techniques under data-driven models can help to solve the problems of feature extraction, complex pattern recognition, large-scale data processing and vulnerability detection in malware detection, thus greatly improving the accuracy and efficiency of malware detection. Therefore, a software security detection model based on optimised deep learning is proposed. Firstly, the analysis of the features yields that these features have a differentiated percentage of malicious and benign samples as well as their importance. Therefore, permission features, API call features, and opcode call sequence features, which have a greater contribution to distinguish malicious software from benign software, are selected to generate the training set. Then, in order to improve the detection rate of low-frequency attack types in the classification task, Adaptive Synthetic Sampling (ADASYN) technique is used for data balancing process. Next, the features are fed into an optimised multi-granularity scanning module to extract features with strong representational properties. Finally, XGBoost is added as a base learner in the deep forest algorithm, while the random inactivation algorithm is introduced, and the feature vectors are fed into the cascade forest module to achieve layer-by-layer training and complete malware classification. Simulation tests are conducted through 13226 malicious samples and 13520 benign samples. The experimental results show that the malware detection based on the improved deep forest is more effective than the traditional model, and the accuracy rate reaches 92.04 %.
Keywords: malware; software security; deep learning; classification detection; deep forests

1. **Introduction.** Artificial Intelligence (AI) technologies play an important role in detecting and preventing malware. First, AI can analyse large-scale datasets through machine learning algorithms to identify characteristic patterns of malware, thus helping security researchers to discover new types of malware more quickly. Second, AI can also be used for behavioural analysis to prevent the spread and damage of malware by monitoring the behavioural patterns and network activities of applications to detect abnormal

behaviours and provide timely warnings [1, 2]. In addition, AI can be used for automated malicious code identification and virtualisation techniques to help quarantine and remove malware [3].

Deep learning techniques in the field of artificial intelligence have many advantages in detecting specific types of malware. Deep learning models can automatically learn and extract features associated with specific types of malware from large-scale datasets [4, 5]. For specific types of malware such as ransomware or spyware, these features can be complex and difficult to extract manually, and deep learning can automatically discover and utilise these features [6], improving the accuracy and efficiency of detection. There may be complex non-linear relationships between the features of malware, and deep learning models can better capture and identify these complex relationships, which can more accurately distinguish between specific types of malware and normal applications. Deep learning techniques can save a lot of training time and resources by applying models trained on a specific type of malware detection to other related types of malware detection through migration learning methods [7, 8]. Deep learning models can process large-scale and real-time data more efficiently, and therefore are better adapted to the rapid evolution and large-scale spread of malware.

Deep learning techniques under data-driven models can help to solve the problems of feature extraction, complex pattern recognition, large-scale data processing and vulnerability detection in malware detection, thus greatly improving the accuracy and efficiency of malware detection. Therefore, this work proposes a software security detection model based on optimised deep forest. The deep forest model can effectively fuse multiple types of data, such as static and dynamic analysis features, API call sequences, and so on. By integrating multimodal information, the deep forest model can capture malware features more comprehensively, thus improving detection accuracy.

1.1. **Related work.** Machine learning-based malware detection is a direction that has received a lot of attention in the current research field, including feature engineering [9, 10], traditional machine learning algorithms [11] and deep learning methods [12].

Masabo et al. [13] proposed a malware family classification method based on feature engineering. By analysing the features of malware samples, the researchers extracted the common features of malware families, including features such as file name, file size, signature, and so on. Using these features to classify malware samples can better identify malware of different families and improve the accuracy and efficiency of malware detection. Daeef et al. [14] proposed a method to classify malware samples by considering multiple features of malware samples together. The researchers use the techniques of feature selection and feature fusion to extract the key features of malware samples, and effectively classify and identify malware by constructing a classification model. Gibert et al. [15] studied the feature engineering analysis and detection methods of malware samples, and conducted in-depth analysis of malware samples from multiple dimensions such as file structure, binary code, and behavioural features. The researchers proposed a deep learning-based malware detection method, which uses neural networks to learn and model the features of malware, and achieved better results in the field of malware detection.

Kayode-Ajala [16] provides a comprehensive survey of traditional machine learning techniques used for malware detection, including decision trees, random forests, Support Vector Machines (SVMs), and K-Nearest Neighbors (KNN). It also discusses the limitations and challenges of using traditional machine learning techniques for malware detection and suggests future research directions.

Lee et al. [17] introduced the application of genetic algorithms in feature selection for malware detection. The advantages of genetic algorithms are investigated, like the genetic

search strategy helps to avoid local optimal solutions to determine the optimal subset of features. The results demonstrated that genetic feature selection is significantly effective in improving the accuracy and reducing the number of features for malware detection as compared to other feature selection methods. Wüchner et al. [18] proposed a method for mining malware behaviour using graph patterns for feature engineering. The use of graph patterns to obtain malware behavioural features can reveal more complex malicious behavioural patterns than traditional signature or behavioural sequence based methods, thus improving the accuracy and scope of detection.

Jian et al. [19] proposed to detect malware by converting binary files into greyscale images and then using deep neural networks. The advantage of this approach is that it can detect malware without needing to know its specific structure or behavioural patterns in advance, but it may be affected by malware authors by adding invalid opcodes or other evasion techniques. Li et al. [20] explored the effectiveness of applying recurrent neural networks to malware classification. The authors used recurrent neural networks to process malware sequence data (e.g., API call sequences) to identify malicious behaviour patterns. Recurrent neural networks are particularly suitable for processing time-series data and can be used to capture changes in malware behaviour during execution.

1.2. **Motivation and contribution.** Deep forest algorithm has some advantages in malware detection compared to recurrent neural networks. Deep forest algorithm is more suitable to deal with high-dimensional sparse data and can perform feature selection efficiently, while RNN is suitable to deal with time series data [21, 22]. In malware detection, the samples are often high-dimensional sparse features. The random forest in deep forest algorithm can give the importance of each feature [23, 24], which is helpful to understand the model, while RNN is black-box model, which is less suitable for interpretation. Deep forest algorithm is faster to train and easier to parallelise.

Therefore, this work proposes the application of deep forest algorithms to malware detection tasks in order to improve the accuracy of classification and identification. The main innovations and contributions of this work include: (1) Since a single behavioural feature is not enough to describe all the behaviours of an application, this paper first analyses a large number of malicious and benign samples, and comes up with three types of static features that have a high rate of contribution to distinguishing between malicious and benign software, i.e., privilege features, API call features, and opcode call features. These three types of features are used to generate the training set for subsequent deep forests. (2) To improve the detection rate of low-frequency attack types in the classification task, Adaptive Synthetic Sampling (ADASYN) technique [25] is used for data balancing. (3) Adjust the multi-granularity scanning module of deep forest, and add XGBoost as the base learner in the cascade forest module to increase the diversity of learners. The stochastic deactivation algorithm is introduced to save computational resources and increase the speed of model training.

2. Characterisation and classification of software security testing.

2.1. **Malware characteristics.** For malware detection, we first analyse the existing malware to find out their main characteristics and categories. Our knowledge of these behavioural characteristics and classifications will help us in our subsequent detection efforts.

Malware has a variety of characteristics, including covertness, destructiveness, and theft of user information. First of all, malware is stealthy in that it can be hidden in mobile phone applications in various ways that are difficult to be detected by users. For example, it may disguise itself as a common application or hide behind other applications and run on the mobile phone without the user's consent. Secondly, malware is destructive

and can cause serious damage to the mobile phone system, including deleting important files and tampering with system settings. Once infected, the mobile phone may become slow, crash frequently, or even become unusable. Most importantly, malware also has the characteristic of stealing users' information, and it can obtain all kinds of users' private information on their mobile phones.

There are many characteristics of malware and the following are some of the common ones [26]:

(1) Permission abuse: Malware may request too many permissions to access a user's device, including access to sensitive information and features such as SMS, contacts, location, camera, and so on.

(2) Ad Bombing: Certain malware will display a large number of advertisements on the user's device in order to make profit. These adverts may be displayed in the form of pop-ups, interstitials, browser redirects, and so on.

(3) Data theft: Malware may steal users' personal information, such as account passwords, bank card information, address books, etc., and send this information to the attacker.

(4) Cheating: Certain malware may cheat by modifying game data, simulating clicks, etc. to gain an undue advantage.

(5) Backdoor Functions: Malware may gain remote control of user devices through backdoor functions, which allows attackers to access and control user devices at any time.

(6) Battery and performance drain: Some malware can run continuously in the background, draining the device's battery and performance resources, resulting in slowdowns and reduced battery life.

(7) Pop-up spoofing: Certain malware will disguise itself as legitimate applications or system prompts through pop-up windows to lure users to click on advertisements or perform dangerous behaviours.

(8) Superuser privilege acquisition: Some advanced malware attempts to gain superuser privileges (root) so that the attacker can take full control of the device and bypass some system restrictions.

2.2. **Classification of malware.** To detect malware, first define what is malware? What are the types of malware available? As defined by Microsoft, malware is any software designed to cause damage to an individual computer, server, or computer network. Malware is code that runs on an operating system to compromise the confidentiality, integrity, and effectiveness of the system by exploiting system vulnerabilities or creating vulnerabilities. Following are some common classifications of Android malware [27]:

(1) Spyware: This type of malware is designed to silently monitor a user's mobile phone activity, including stealing text messages, call logs, location information and other sensitive data.

(2) Trojan: This malware disguises itself as a legitimate application, and once the user installs it, it performs malicious operations in the background, such as stealing information and damaging the system.

(3) Adware: This type of malware mainly makes profits by displaying annoying advertisements and may cause slowdowns or other problems with mobile phones.

(4) Ransomware: This malware encrypts the user's data or restricts the user's access to the mobile phone, and then blackmails the user into paying a ransom to unlock the phone or decrypt the data.

(5) Downloader: This malware will download and install other malware without the user's knowledge, thus expanding the scope of infection.

(6) Rootkit: This kind of malware can hide inside the system, making it difficult for users to detect and remove, thus lurking in the mobile phone system for a long time.

3. Software sample characterisation and optimisation.

3.1. **Permission characterisation.** Permission mechanism is used as an access control provided to the application by the Android system. Androguard toolkit is a Python based static reverse analyser which is very scalable. It will map the files, methods and classes etc. obtained after unzipping the APK file to python objects. A separate Python file in the toolkit provides a rich set of tools to accomplish different functions, which are used to manipulate these objects to extract the various information needed. We use `androlyze.py` from the Androguard toolkit to execute commands to analyse APK files in an interactive environment. As you can see in the Androguard source code, the class that creates the APK file object is the `androguard.core.bytecodes.apk.APK` class, which has access to all the elements in the APK file. Among other things, the class gets the manifest file using `get_AndroidManifest()`, uses `get_permissions()` to get the permissions information in the application, and uses `get_apis()` to get API call information. Use `get_package()` to get the package name.

Firstly, three objects, APK file, DEX file and analysis results, are initialised in the interactive environment. Then, the AndroidManifest.xml files in the benign and malicious application samples are processed using `get_permissions()` in the Androguard toolkit to obtain their permission information, respectively. The comparison reveals that some of the permissions that account for a large difference in the percentage of permission features in the malicious and benign samples are shown in Figure 1. After the above analysis, we choose the above 153 permission features as the focus features as they are significantly different in malicious and benign samples. Their chi-square values will be further calculated for validation in the later feature optimisation stage to arrive at the final set of permission features, which are then feature vectorised.

3.2. **API call characterisation.** There is a direct relationship between API calls and the behaviour of an application. Because applications use system resources through API calls to achieve application-specific functionality, system call characteristics can more accurately characterise the behaviour of an application. In order to more accurately characterise the malware behaviour, we statically obtain and analyse these sensitive API calls in smali files. In this paper we focus on API call profiles that refer to function interfaces provided by the Android system. It is through these function calls that the application accesses and obtains system resources, such as obtaining the user's geolocation, accessing contacts, accessing storage, and so on. We write Python scripts to traverse the classes.dex files in malicious and benign samples respectively and decompile them into smali format files. Then, the `get_apis0` method in Androguard toolkit is used to get the API call information, including API name, parameters and return value. After traversing the smali file, the number of API calls for some malicious samples and benign samples are counted as shown in Figure 2. Through the analysis, 33 common high-risk API call features are obtained.

3.3. **Opcode call characterisation.** All programs are eventually compiled into machine instructions to command the hardware to work, so similar to API calls, opcodes can also reflect the real behavior of an application. If we use the official instruction set as an alternative instruction base for opcodes, then it will form a more complex sequence of opcodes, which will lead to an excessive amount of feature space data. So in this paper we have used a semantic similarity based compression coding method in smali file to

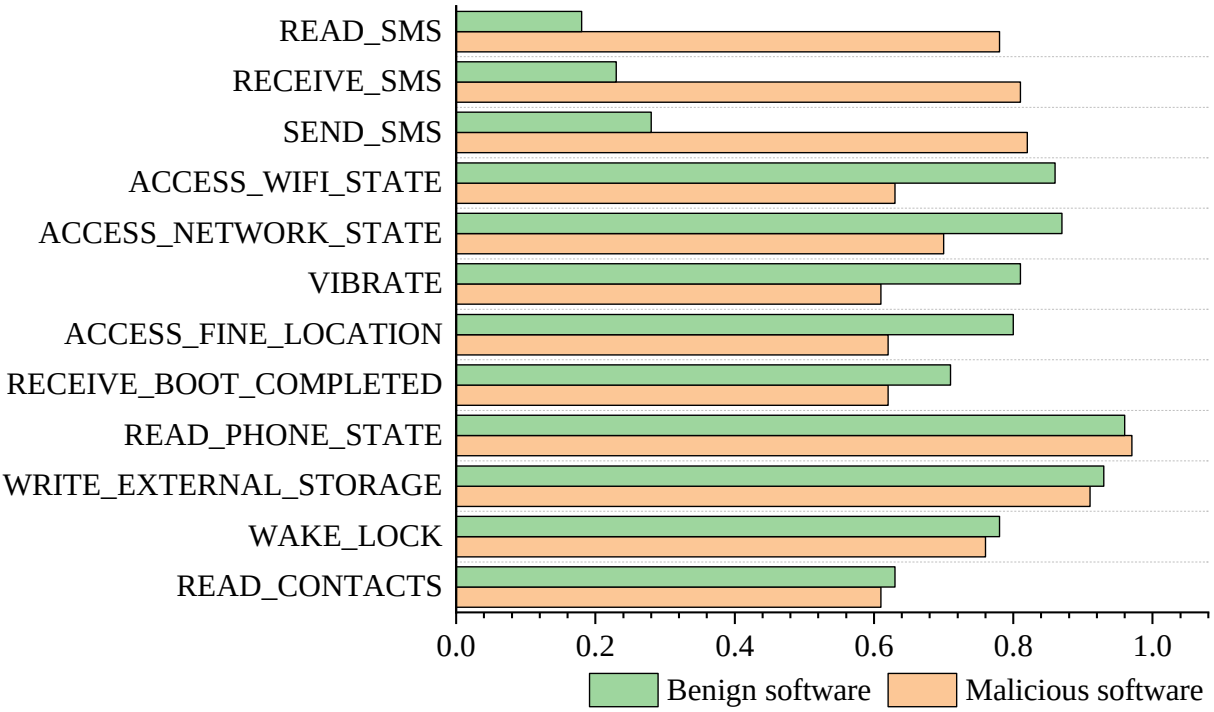

Figure 1. Selected permissions with large differences in permissions characteristics

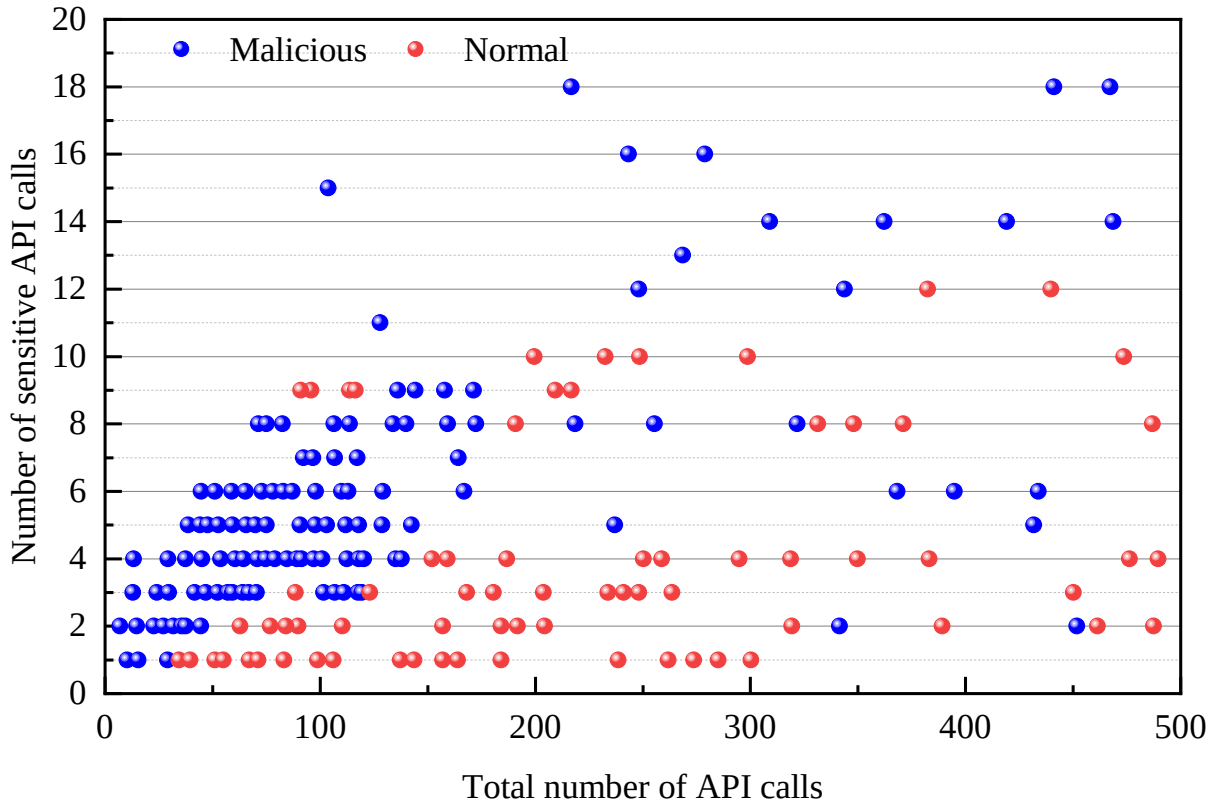

Figure 2. API call statistics

extract the opcode sequence features. The opcode instructions are compressed and coded

based on the similarity between them in order to reduce the amount of data features and redundant information.

3.4. **Feature optimisation.**

3.4.1. *Chi-square test.* The chi-square test is used to optimise the features. Because there is a strong correlation between certain features, as well as no strong correlation between certain features and malware, these meta-surplus features need to be removed during the experiment for the purpose of optimising the features.

The chi-square test is a method commonly used for hypothesis testing based on the distribution of x^2. The central idea is to count the degree of deviation between the actual value of the sample and the theoretical value. The larger the chi-square value, the greater the degree of deviation between the two. The chi-square value is calculated as shown below:

$$x^2 = \sum \frac{(A-T)^2}{T} \tag{1}$$

where the x^2 value indicates the degree of deviation between the real and theoretical values, A is the actual value, and T is the theoretical value.

In testing the correlation between a permission and malware, an independent samples four-cell table is formed as shown in Table 1.

Table 1. Quadratic table of chi-square calculations

	Malware software	**Benign software**	**Total**
$P_m = 1$	A_1	A_2	$A_1 + A_2$
$P_m = 0$	A_3	A_4	$A_3 + A_4$
Total	$A_1 + A_3$	$A_2 + A_4$	$A_1 + A_2 + A_3 + A_4$

Then, 153 permissions provided by Android were calculated.

$$x^2 = \frac{N(A_1A_4 - A_2A_3)^2}{(A_1 + A_3)(A_2 + A_4)(A_1 + A_2)(A_3 + A_4)} \tag{2}$$

where x^2 value denotes the correlation between a permission and malware; $N = A_1 + A_2 + A_3 + A_4$; A_1, A_2, A_3, and A_4 denote the statistics of the four permission feature sets.

The filtering is done by calculating the chi-square value of each permission feature (the permission features with chi-square value $x^2 \leq 0.01$), i.e., filtering out whether they are strongly correlated with malware or not, in order to achieve the purpose of dimensionality reduction and optimisation of permission vectors. In this paper, we finally selected 50 system privileges as important privilege features.

3.4.2. *Document word frequency.* Document word frequency is used as a measure of the importance of the opcode sequence. The frequency measure of importance means that the more frequently the feature in question occurs in the benign sample, the more important it is for the benign sample. The document word frequency method is used to represent the correlation between the number of times a feature occurs in a malicious or benign sample and the attributes of the sample as shown below:

$$DF(s,c) = \frac{\log(DF_s)}{\log(N_c)} \tag{3}$$

where DF_s denotes the number of times the feature sequence s occurs in the malicious or benign sample c; and N_c denotes the total number of samples in the malicious or benign

sample. Setting the threshold and selecting those features that have higher correlation with the sample attributes, i.e., selecting features from both malicious and benign samples in a balanced way as much as possible.

3.4.3. *Feature vector generation.* Combined with the above analysis on permission features, the 50 permission features vectorised based on the chi-square test are $X = (s, x_1, x_2, \ldots, x_{50})$. $s \in \{0, 1\}$ is the sample label, 0 means malicious sample, 1 means benign sample. $x_i \in \{0, 1\}$ is the feature label, 0 means the feature is not included in the APK sample, 1 means the feature is included in the APK sample.

The 33 sensitive API calls are vectorised into a 33-dimensional vector $(s, a_0, a_1, \ldots, a_{33})$, with $s \in \{0, 1\}$ as the sample labels, and 0 denoting the malicious samples and 1 denoting the benign samples. Each dimension a_i represents the number of times an application has called the API.

The 44 groups of opcode features obtained after compression coding are vectorised, i.e., a 44-dimensional vector of opcode calling sequences is generated, $P = (S, O_1, O_2, O_3, \ldots, O_i, \ldots, O_{44})$, where $s \in \{0, 1\}$ is the sample label, 0 denotes a malicious sample, 1 denotes a benign sample, O_i denotes the number of occurrences of the opcode i. If the opcode i does not appear, then O_i is zero. The importance of an opcode sequence is then evaluated by combining the document word frequency method proposed above. We simplify the calculation by summing the number of occurrences of each operand, i.e., the larger the sum of $O_1 + O_2 + \cdots + O_i + \cdots + O_{44}$, the larger the weight of the sequence.

4. **Software Security Detection Model Based on Improved Deep Forest.** On the basis of the above feature optimisation, this paper proposes an Improved Deep Forest Classification Model (IDFCM). The detection of malware by the IDFCM model consists of two phases: feature vector extraction and classification detection.

4.1. **Oversampling-based data balancing.** Inevitably, there is a low-frequency attack type in many malware. The sample size of this attack type is very small, so it is difficult to be detected. In order to improve the detection accuracy and recall of low-frequency attack types, an oversampling technique is proposed to be added to the model in order to balance the dataset. Compared to the traditional oversampling technique which simply replicates a few classes of samples, the key idea of Synthetic Minority Over-sampling Technique (SMOTE) lies in the introduction of synthetic samples, the implementation of which is shown in Figure 3. The synthetic method of SMOTE is as follows:

$$x_n = x_i + \beta \times (x_j - x_i) \tag{4}$$

where β is a random number in the interval $(0, 1)$.

For SMOTE, all minority samples are the same and do not care about the category information of the nearby samples, which leads to the phenomenon of sample aliasing and thus affects the effectiveness of classification. In contrast to SMOTE, which creates a new instance for each minority sample, Borderline SMOTE identifies only the borderline minority samples, and then selects its K nearest neighbours based on the borderline minority samples to synthesize a new instance. Since Borderline SMOTE only randomly selects the boundary minority class samples for new sample synthesis, it avoids generating too much data inside the minority class and also eliminates the interference of noise. Therefore compared to SMOTE, Borderline SMOTE has better results.

In this paper, ADASYN technique is used for data balancing, the most important feature of ADASYN is that it can calculate the learning difficulty of minority class samples compared to SMOTE and Borderline SMOTE. For the minority class samples that are

more difficult to learn, more new data are synthesised by performing weighting. First, the number of new samples to be synthesised is calculated:

$$N = (l - s) \times \beta \tag{5}$$

where l represents the number of majority class samples, s represents the number of minority class samples, and $\beta \in [0, 1]$ represents the random number.

Then, calculate the percentage of majority class samples in the K nearest neighbours of the minority class samples:

$$r_i = \Delta_i \,/\, K \tag{6}$$

where Δ_i is the number of majority class samples in the K nearest neighbourhood, $i = 1, 2, 3, \ldots, s$.

Standardise on r_i:

$$\hat{r}_i = \frac{r_i}{\sum_{i=1}^{s} r_i} \tag{7}$$

The number of new samples to be generated is calculated for each minority sample according to its learning difficulty:

$$n = \hat{r}_i \times N \tag{8}$$

Finally, for each minority class sample, n new samples are generated by interpolation:

$$s_i = x_i + (x_{zi} - x_i) \times \lambda \tag{9}$$

where x_{zi} is the K nearest-neighbour sample of x_i, and λ is a random number of $[0, 1]$.

4.2. **Improved deep forests.**

4.2.1. *XGBoost.* XGBoost employs a second-order Taylor expansion for objective function optimisation, which allows for better handling of outliers and automatic treatment of missing values without the need to manually process them in preprocessing, reducing their impact on the model [28].

The predicted values for the i-th sample of XGBoost are as follows.

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in F \tag{10}$$

where K represents the number of trees; F represents the set of CART trees; x_i represents the feature vector of the i-th node; and f_k represents the structure corresponding to the k-th tree and the weight score w of its leaves.

The loss function for XGBoost is as follows.

$$L = \sum_{i=1}^{n} l\big(y_i, \hat{y}_i\big) \; + \; \sum_{k=1}^{K} \Omega(f_k) \tag{11}$$

$$\Omega(f) = \gamma\, T \; + \; \tfrac{1}{2}\, \lambda \sum_{j=1}^{T} w_j^2 \tag{12}$$

By adjusting the weights of the new learner and weighting the training, the objective function is iteratively approximated to be minimised. The objective function for the t-th iteration is as follows.

$$L^{(t)} = \sum_{i=1}^{n} l\big(y_i,\ \hat{y}_i^{(t-1)} + f_t(x_i)\big)\ +\ \Omega\big(f_t\big) \tag{13}$$

Approximating the objective function by a second-order Taylor expansion at $f_t = 0$, we have the objective X_{obj} approximated as follows:

$$\begin{aligned} X_{\text{obj}} &\approx \sum_{i=1}^{n} \Big[g_i\, f_t(x_i) + \tfrac{1}{2}\, h_i\, f_t^2(x_i) \Big] + \Omega\big(f_t\big) \\ &\approx \sum_{i=1}^{n} \Big[g_i\, f_t(x_i) + \tfrac{1}{2}\, h_i\, f_t^2(x_i) \Big] + \gamma\, T + \tfrac{1}{2}\, \lambda \sum_{j=1}^{T} w_j^2 \\ &\approx \sum_{j=1}^{T} \Big[\big(\sum_{i\in I_j} g_i\big) w_j + \tfrac{1}{2} \big(\sum_{i\in I_j} h_i + \lambda\big) w_j^2 \Big] + \gamma\, T \end{aligned} \tag{14}$$

where g_i is the first-order derivative of the loss function and h_i is the second-order derivative.

Next, define $G_j = \sum_{i\in I_j} g_i$, $\quad H_j = \sum_{i\in I_j} h_i$, the optimal leaf weight w_j and objective function value are as follows.

$$w_j = -\frac{G_j}{H_j + \lambda} \tag{15}$$

$$X_{\text{obj}} = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j}{H_j + \lambda} + \gamma\, T \tag{15}$$

The loss function should be continuously approximated towards minimisation, and the subtree should be divided using a greedy algorithm while enumerating the possible splits. Each time a new split is added to an existing leaf node, and the gain obtained is calculated and maximised. The gain L_{Gain} is as follows.

$$L_{\text{Gain}} = \frac{1}{2} \Big[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)}{H_L + H_R + \lambda} \Big] - \gamma \tag{16}$$

4.2.2. *Multi-granularity scanning adjustment.* The multi-granularity scanning link of the traditional deep forest refers to the sliding convolution kernel feature extraction of CNN networks, which can effectively enhance the characterisation ability by scanning the features through a sliding window and inputting them into the cascade forest after conversion. However, due to the characteristics of malware feature data, the relationship between different features is not as obvious as neighbouring pixels in the image domain.

Therefore, for the malware detection problem, it is necessary to consider various possibilities of combinations among features. In this paper, the multi-granularity scanning module is adapted according to the characteristics of malware data, and different sizes of selectors can be chosen to extract features as completely as possible. Pre-processing before transferring to the cascade forest module can effectively explore the potential connection between features and effectively enhance the learning ability of network security posture element data. A comparison of the adjusted multi-granularity scan with the traditional deep forest is shown in Figure 3.

In Improved Multi-Granularity Scanning, class vectors with n dimensions are generated based on each instance. By stitching all the class vectors together, a long feature sequence containing probabilistic information can be obtained eventually. It is also possible to

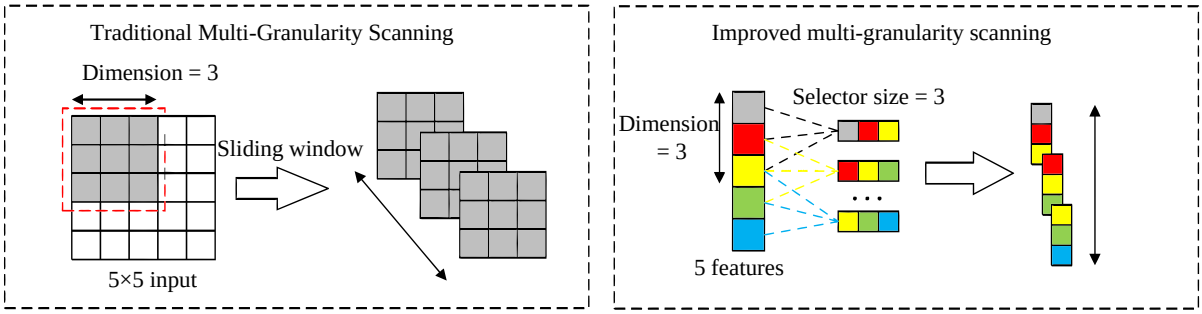

Figure 3. Comparison of two multi-granularity scans

generate vectors with different granularities by choosing selectors with multiple sizes, which makes the feature extraction better.

4.2.3. *Improved cascading.* Deep forests are inspired by the idea of layer-by-layer processing in deep neural networks, and each layer is composed of random forests and completely random forests to achieve the feature information transfer and processing at each level.

The performance of deep forest depends on the accuracy of each base learner and the variability between them. In this paper, XGBoost is added as a base learner in each layer of the cascade forest to improve the diversity of the deep forest. Due to the increase in the number of deep forest layers, overfitting may occur, and due to the massive and high-dimensional characteristics of cybersecurity situational data, a large amount of sample data will be generated in the multi-granularity scanning module, resulting in the consumption of a large number of computational resources during training and classification. In order to solve the above problems, in the cascade forest module we join the Dropout algorithm, randomly and temporarily shutting down part of the node's data transmission, in order to enhance the performance of the deep forest. The Dropout function d_t is defined as follows:

$$d_t = [v_t^1, v_t^2, \ldots, v_t^N]_t^N = \begin{cases} 1, & p_i > p, \\ 0, & p_i < p, \end{cases} \tag{17}$$

where N denotes the number of channels; t denotes the number of cascade levels; p_i denotes the probability of the i-th channel Dropout; and p is a hyperparameter controlling whether the channel is open or not.

5. **Experimental results and analyses.**

5.1. **Experimental samples and environment configuration.** Most of the available malware samples come from researchers as well as from sharing on websites. Many researchers also tend to write their own malware. VirusShare website is a personal project website that provides malware samples for researchers. We have collected a total of 12,083 malware samples through this platform. In Contagio, the sharing site for the "Android Malware Genome Project" project, we obtained a total of 1,143 malicious samples. The total number of malicious samples obtained is 13,226, as shown in Table 2.

Table 2. Sources of malicious samples

Number of malicious samples	Sample Sources	Web address
12083	100	https://virusshare.com
1143	14	http://contagiominidump.blogspot.com/

We obtained benign samples through a variety of channels, mainly writing shell scripts and using APKPure download tool to download a total of 13,520 benign samples from Sina website, Automotive website, Huawei, and Xiaomi app malls, as shown in Table 3. Therefore, the malicious samples in the experiment are 13,226, and the benign samples are 13,520.

Table 3. Sources of good samples

Number of benign samples	Sample Sources	Web address
10000	Sina Website	https://www.sina.com.cn/
1320	Motor Home Website	https://www.autohome.com.cn/
1200	Huawei App Store	https://consumer.huawei.com/cn
1000	Xiaomi App Store	https://app.mi.com/

IDFCM-based malware detection is experimented using 5-fold cross-validation, and the specific simulation environment configuration is shown in Table 4.

Table 4. Simulation environment configuration

Simulation environment	Specific configuration
Operating system	Windows 10 Professional 20H2 (19042.1466)
CPU	Intel® Core(TM) i5-10400 CPU @ 2.90GHz
Random access memory (RAM)	16.0 GB (DDR4 2666 MHz, 8 GB × 2)
Hard drive	500 GB
Display card (computer)	NVIDIA GeForce RTX 2070 8 GB
Development language	Python 3.9.12

5.2. **Experimental result.** The IDFCM was compared with the DFCM in each evaluation metric. In the experiments, each classification detection model is evaluated in terms of four metrics (recall, precision, accuracy and F1-score). The experimental performance metrics of each model are shown in Figure 4.

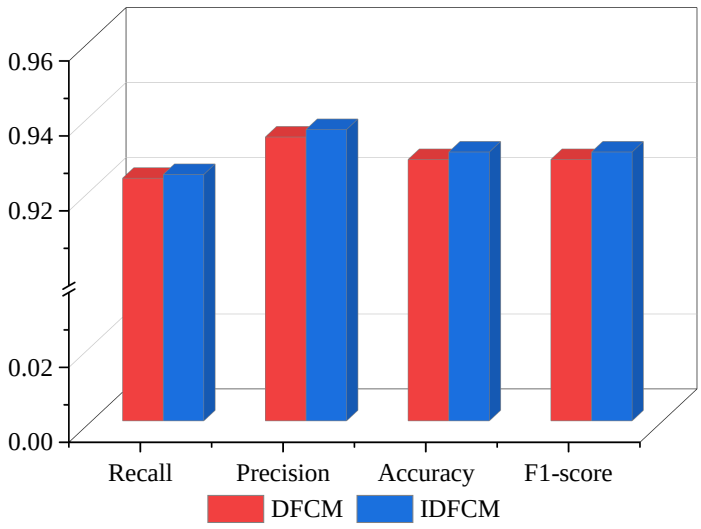

Figure 4. Overall performance evaluation

In terms of accuracy, the IDFCM classification model is more accurate than the DFCM classification model in malware detection with 92.04%. In terms of recall and precision, the

IDFCM classification model is more effective. In terms of F1-score, Bi-LSTM classification model is more stable.

6. **Conclusion.** This work proposes to apply deep forest algorithm to malware detection task in order to improve the accuracy of classification and identification. Firstly, a large number of malicious and benign samples are analysed to derive 3 types of static features that contribute highly to the distinction between malicious and benign software, i.e., permission features, API call features, and opcode call features. These 3 types of features are used to generate the training set required for subsequent deep forests. In order to improve the detection rate of low-frequency attack types in the classification task, ADASYN technique is used for data balancing process. Then, the multi-granularity scanning module of deep forest is adapted to add XGBoost as a base learner in the cascade forest module to increase the learner diversity. The stochastic deactivation algorithm is introduced to save computational resources and increase the speed of model training. Finally, 13226 malicious samples are collected from VirusShare website and Contagio website. The benign samples from several business-like websites mobile phones are 13520. The experimental results show that IDFCM has good classification accuracy.

REFERENCES

[1] S. Töyssy and M. Helenius, "About malicious software in smartphones," *Journal in Computer Virology*, vol. 2, pp. 109–119, 2006.

[2] J. Singh and J. Singh, "Detection of malicious software by analyzing the behavioral artifacts using machine learning algorithms," *Information and Software Technology*, vol. 121, 106273, 2020.

[3] B. K. Mishra and N. Jha, "Fixed period of temporary immunity after run of anti-malicious software on computer nodes," *Applied Mathematics and Computation*, vol. 190, no. 2, pp. 1207–1212, 2007.

[4] H. Shi, J. Mirkovic, and A. Alwabel, "Handling anti-virtual machine techniques in malicious software," *ACM Transactions on Privacy and Security*, vol. 21, no. 1, pp. 1–31, 2017.

[5] M. Rahmatian, H. Kooti, I. G. Harris, and E. Bozorgzadeh, "Hardware-assisted detection of malicious software in embedded systems," *IEEE Embedded Systems Letters*, vol. 4, no. 4, pp. 94–97, 2012.

[6] İ. A. Doğru and Ö. KİRAZ, "Web-based android malicious software detection and classification system," *Applied Sciences*, vol. 8, no. 9, 1622, 2018.

[7] F.-Q. Li, S.-L. Wang, A. W.-C. Liew, W. Ding, and G.-S. Liu, "Large-scale malicious software classification with fuzzified features and boosted fuzzy random forest," *IEEE Transactions on Fuzzy Systems*, vol. 29, no. 11, pp. 3205–3218, 2020.

[8] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic analysis of malicious code," *Journal in Computer Virology*, vol. 2, pp. 67–77, 2006.

[9] T.-Y. Wu, H. Li, S. Kumari, and C.-M. Chen, "A Spectral Convolutional Neural Network Model Based on Adaptive Fick's Law for Hyperspectral Image Classification," *Computers, Materials & Continua*, vol. 79, no. 1, pp. 19–46, 2024.

[10] T.-Y. Wu, A. Shao, and J.-S. Pan, "CTOA: Toward a Chaotic-Based Tumbleweed Optimization Algorithm," *Mathematics*, vol. 11, no. 10, 2339, 2023.

[11] T.-Y. Wu, H. Li, and S.-C. Chu, "CPPE: An Improved Phasmatodea Population Evolution Algorithm with Chaotic Maps," *Mathematics*, vol. 11, no. 9, 1977, 2023.

[12] M. Alazab, "Profiling and classifying the behavior of malicious codes," *Journal of Systems and Software*, vol. 100, pp. 91–102, 2015.

[13] E. Masabo, K. S. Kaawaase, J. Sansa-Otim, J. Ngubiri, and D. Hanyurwimfura, "Improvement of malware classification using hybrid feature engineering," *SN Computer Science*, vol. 1, pp. 1–14, 2020.

[14] A. Y. Daeef, A. Al-Naji, and J. Chahl, "Features Engineering for Malware Family Classification Based API Call," *Computers*, vol. 11, no. 11, 160, 2022.

[15] D. Gibert, J. Planes, C. Mateu, and Q. Le, "Fusing feature engineering and deep learning: A case study for malware classification," *Expert Systems with Applications*, vol. 207, 117957, 2022.

[16] O. Kayode-Ajala, "Applying Machine Learning Algorithms for Detecting Phishing Websites: Applications of SVM, KNN, Decision Trees, and Random Forests," *International Journal of Information and Cybersecurity*, vol. 6, no. 1, pp. 43–61, 2022.

[17] J. Lee, H. Jang, S. Ha, and Y. Yoon, "Android malware detection using machine learning with feature selection based on the genetic algorithm," *Mathematics*, vol. 9, no. 21, 2813, 2021.
[18] T. Wüchner, A. Cisłak, M. Ochoa, and A. Pretschner, "Leveraging compression-based graph mining for behavior-based malware detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 1, pp. 99–112, 2017.
[19] Y. Jian, H. Kuang, C. Ren, Z. Ma, and H. Wang, "A novel framework for image-based malware detection with a deep neural network," *Computers & Security*, vol. 109, 102400, 2021.
[20] S. Li, Q. Zhou, and W. Wei, "Malware detection with directed cyclic graph and weight merging," *KSII Transactions on Internet and Information Systems*, vol. 15, no. 9, pp. 3258–3273, 2021.
[21] T. Zhou, X. Sun, X. Xia, B. Li, and X. Chen, "Improving defect prediction with deep forest," *Information and Software Technology*, vol. 114, pp. 204–216, 2019.
[22] L. V. Utkin and M. A. Ryabinin, "A siamese deep forest," *Knowledge-Based Systems*, vol. 139, pp. 13–22, 2018.
[23] L. Sun, Z. Mo, F. Yan, L. Xia, F. Shan, Z. Ding, B. Song, W. Gao, W. Shao, and F. Shi, "Adaptive feature selection guided deep forest for covid-19 classification with chest ct," *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 10, pp. 2798–2805, 2020.
[24] J. Cheng, M. Chen, C. Li, Y. Liu, R. Song, A. Liu, and X. Chen, "Emotion recognition from multichannel EEG via deep forest," *IEEE Journal of Biomedical and Health Informatics*, vol. 25, no. 2, pp. 453–464, 2020.
[25] Z. Hu, L. Wang, L. Qi, Y. Li, and W. Yang, "A novel wireless network intrusion detection method based on adaptive synthetic sampling and an improved convolutional neural network," *IEEE Access*, vol. 8, pp. 195741–195751, 2020.
[26] P. Maniriho, A. N. Mahmood, and M. J. M. Chowdhury, "A study on malicious software behaviour analysis and detection techniques: Taxonomy, current trends and challenges," *Future Generation Computer Systems*, vol. 130, pp. 1–18, 2022.
[27] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on opcode patterns," *Security Informatics*, vol. 1, no. 1, pp. 1–22, 2012.
[28] A. Ogunleye and Q.-G. Wang, "XGBoost model for chronic kidney disease diagnosis," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 17, no. 6, pp. 2131–2140, 2019.