International Journal on Advanced Science Engineering Information Technology

Vol.15 (2025) No. 3
ISSN: 2088-5334

A Deep Learning Approach to Malware Detection: Leveraging Multilayer Perceptron for Permission-Based

Ameerah Muhsinah Jamil [a], Mohd Faizal Ab Razak [a,1], Sharfah Ratibah Tuan Mat [b], Mahir Pradana [c], Deden Witarsyah [d,2]

[a] *Faculty of Computing, University Malaysia Pahang, Gambang, Kuantan, Pahang, Malaysia*
[b] *Department of Mathematics and Computer Science, Politeknik Sultan Haji Ahmad Shah, Malaysia*
[c] *Department of Business Administration, Telkom University, Bandung, Indonesia*
[d] *Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, Malaysia*
Corresponding authors: [1]faizalrazak@umpsa.edu.my; [2]deden@uthm.edu.my

***Abstract*— The active growth of social networking worldwide has encouraged the emergence of malware that threatens such devices. Continuously researching according to malware threats has been accomplished to prevent the malware spread. Yet, malware attack continues to** change **and occur in very large numbers that requiring better solutions. In this paper, we proposed a Multilayer perceptron, a type of deep learning approach to tackle malware attacks focused on permission features. The study conducted eight experiments with 15, 20, 25, and 30 selected features for both algorithms, utilizing a dataset of 10,000 applications—5,000 benign (Androzoo) and 5,000 malicious (Drebin). The detection process involved three phases: data gathering, preprocessing, and classification, employing 10-fold cross-validation. The validation through all the experiments performed in this study achieved the highest accuracy of 98.2% accuracy, though other feature sets exhibited minimal variation in performance. Further dataset analysis revealed that the INTERNET permission was prevalent in 99% of malware samples and 81% of benign applications, highlighting its widespread use. This study underscores the importance of feature selection in Android malware detection and suggests that future research integrate risk assessment to classify and prioritize permission requests. Risk-based analysis could enhance malware detection by systematically evaluating potential security threats, addressing the rapid proliferation of malware. The findings contribute to the ongoing development of robust Android security mechanisms and encourage further research in permission-based threat mitigation strategies.**

***Keywords*—Feature selection; risk assessment; Android malware; permission-based analysis; deep learning.**

Manuscript received 12 Oct. 2024; revised 8 Jan. 2025; accepted 26 Feb. 2025. Date of publication 30 Jun. 2025.

I. INTRODUCTION

Individuals utilize mobile devices for a variety of purposes, similar to how they use desktop computers for activities such as web surfing, online banking, e-commerce, and social networking. These devices have become an essential aspect of human life, gradually replacing personal computers both at home and in professional settings. The global popularity and high demand for mobile devices, coupled with their diverse functionalities, have provided an opportunity for hackers to develop and distribute malicious code targeting mobile devices. These hackers are incentivized to exploit vulnerabilities and spread threats through various means, potentially causing harm to mobile devices and pilfering sensitive information. They can employ tactics such as damaging the device itself and exfiltrating confidential data [1], [2]. The hackers' gain access to mobile device data is often accomplished through techniques like synchronization, buffer overflows, spamming, and phishing, particularly in the context of Android devices [3].

From the statistics, the risk tool (41.24%) was most threatening in 2019, followed by adware (18.71%) and trojan (11.83%). It is stated that the majority of Android malware falls under the category of Trojans. This type of Android malware unintentionally lures users into subscribing to unnecessary services, resulting in significant harm to their mobile devices [4]. Android malware applications are responsible for surreptitiously gathering user account information, subscribing to premium SMS services, and even exploiting hardware vulnerabilities [5].

Mobile devices offer distinct functionalities compared to desktop computers, including SMS messaging, frequent location updates, and widespread accessibility. Additionally,

mobile users tend to centralize their information storage on phones due to their convenience and swift accessibility. Given their popularity and impressive capabilities, mobile devices have become a primary target for malicious activities. For instance, around 21 million Android devices fell victim to infection during installation from the Google Play Store [6]. In response to this threat, Google has outlined protective measures to help Android users avoid malware attacks.

Google has implemented permission-based features as a central security mechanism for Android users [7]. These permission features are a component of Google's Android malware detection strategy, designed to strengthen the defense against malware infections in the Android ecosystem. Users are empowered to scrutinize the permissions requested and halt the installation process if the permissions appear suspicious or overly permissive. While Android automatically grants some permissions without user confirmation [8], others require explicit user approval depending on the nature of the requested permission, which is classified as either standard or potentially hazardous [9]. Consequently, a thorough evaluation of the malware-related aspects within the application code is imperative to counter and diminish instances of malware breaches effectively.

Static analysis and pattern recognition are used in deep learning malware detection to find complex attack methods. Deep learning models that extract discriminative features from permissions, API calls, and opcode sequences, including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Multi-Layer Perceptrons (MLPs), have been successfully utilized to detect malicious applications. Recent research has demonstrated the efficacy of deep learning in detecting malware in desktop and mobile platforms [10], [11]. Deep learning has improved malware detection accuracy, response times in real-time environments, and achieved better classification results for malware variants, while also reducing computation time [12]-[16]. The implementation of intelligent visualization [17] and the automation model architecture [18]. These developments underscore the growing reliance on deep learning as a vital technique in combating evolving malware threats.

In the field of malware analysis, features, particularly permissions, play a crucial role in classifying malware [19]. Malware analysis involves identifying features within applications that span multiple lines of code. Static analysis extracts these features from numerous lines of code. The comprehensive evaluation of all features and the subsequent selection of pertinent ones from these lines is a stringent task, as security analysts are obligated to initially assess all malware and benign applications to pinpoint relevant features before identifying distinctive attributes or components of malware or benign applications [20]. This highlights the need to reduce machine learning duration by removing noisy and irrelevant data [21], which in turn leads to an effective detection method [22].

An additional security mechanism designed to address malware infections involves the use of an intrusion detection system (IDS). This can take the form of hardware, software, or a hybrid solution, and its purpose is to monitor the activity within a network or system, aiming to identify indicators of malicious intent. This technology is dedicated to assessing the activities within networks and systems, aiming to uncover potential vulnerabilities that could be exploited against a computer system. Furthermore, security analysts employ two static and dynamic analyses. Nonetheless, these approaches prove inadequate in cases where attackers employ techniques to evade detection, such as polymorphism and obfuscation, which complicates the process of risk assessment and Android malware identification. As a result, a thorough examination of permission features and the implementation of an IDS in Android malware identification are undertaken through the utilization of a static analysis methodology.

The key contributions of the study are as follows:

- The experiment was practiced on 96,074 datasets respectively. The samples were retrieved from Drebin for malware and Androozo for benign.
- The experiment applied the permission features extracted from AndroidManifest.xml in malware detection.
- The malware detection applied the Information Gain algorithm for feature optimizations.

The remainder of this paper is organized as follows. Section 2 discusses the approach and flow of malware detection, the general model, and malware detection for feature selection. Section 3 describes the evaluation process and findings of the experiment. Section 4 highlights the paper's challenges and emerging trends, as well as recommendations for future research.

II. Materials and Methods

The model of the primary system is visually shown in Figure 1. The model is divided into three phases: dataset collecting, pre-processing, and detection. These phases are intricately interconnected. The initial step involves obtaining permissions from both benign and malicious applications. This entails the decompilation of the .apk files, followed by the extraction and filtration of permissions. These gathered permissions are then compiled and saved in a readable .arff format. This file contains a comprehensive set of attributes about functionality, thoughtfully refined to enhance efficiency by eliminating extraneous noise and irrelevant elements. To definitively identify malware, a Bayesian classifier is employed, facilitating the classification of both malware and benign instances.

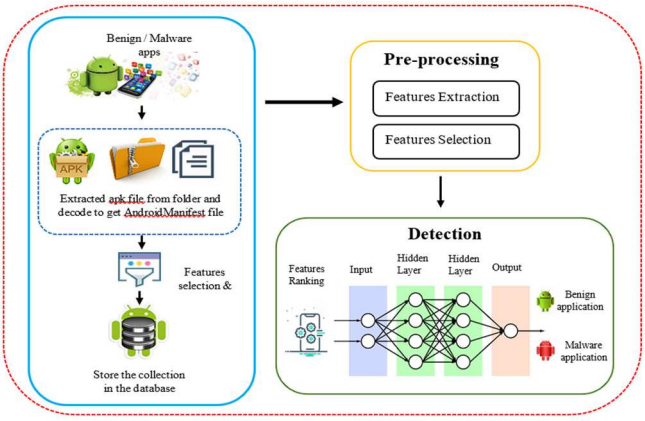

Fig. 1 General model for a Bayesian classifier

Dataset collection, pre-processing (data extraction), feature refinement, and detection are the steps in the Android malware identification process, as depicted in the above

figure. The datasets used in this research comprised a total of 96,074 samples. The subsequent phase in the Android malware detection procedure involved using a deep-learning classifier.

A. Dataset Collection Phase

Initially, the samples underwent manual categorization, resulting in the classification of each as either benign or malicious. This classification process for both malware and benign instances occurred alongside the assessment of the Android application's status on VirusTotal. VirusTotal serves as an online platform tailored for virus detection via uploaded files and URLs. The verification of Android malware samples was conducted using VirusTotal, a tool widely adopted within the research community. The dataset samples were used for the feature selection procedure that followed this validation. Table III provides a summary of the top 20 Android malware families found in the 96,074-sample dataset.

TABLE I
DATASET (96,074 SAMPLES)

Samples	Source	Number of datasets
Benign	Androzoo	48820
Malware	Drebin	47254

TABLE II
THE TOP 20 FAMILIES OF ANDROID MALWARE (96,074 SAMPLES)

Features (SHA256)	Virus Total	Family	Total permission
86d3f5efb290a6b06756c3c8c04907b2c137434c8e28a6e6146e0949426cf976	26	Adrd	63
877b1e8455d444a3e422f8926bc957d77bddfcb0b700ee53e438be2dd6bdbd67	24	Generic	63
8b8bb30a8a1f5c2c214dbfc6809027e869479a7aa918fa90f2e29927d8b43991	10	Domob	48
2ac93b967118e65a8bbfdd47191eff63278e2f453c3e336ead40ed19c7fc621d	13	Commplat	46
24909850e15123ea216c0c40819883b6e704fa31d841a752e2b97a92f2677593	15	Skymobi	44
bf328a4fefb6c503b1412f2fa0cf0322f43757385f7cbd051b9ca832b97b630d	16	Skymobi	44
05045fc2b74f8e40a1a27263639c6e81da36071220bb0f56b7d524eeeca5c0fd	13	DroidRooter	43
73d7cc2075e20cd17733b9ded2d6572863c4d2c4a80adf583b046429667bcdb4	10	Dowgin	43
ad2d0fe6cd6dc8a8ff489ffdeb88a8d2ecdc8a8dff82e75c08e2e2770cab93d8	24	Dowgin	43
0c6fe11473ada6ecf8fae9ab2a394385a3ca88838e9ab54ed3bebfce6353a4d6	8	DroidRooter	43
1aa3ac89ae5958c35bb4a9740a8a310867228f2f80a35a6e092b950e6e17f9be	21	Skymobi	42
1ef66d29669d25f22a42a7a5a208f14970ee27da829acbe9ef8d847b6b80f2f9	18	Dowgin	42
a004fac82b918c26b81aec6b6481d0208cc5cbf9734a229dd1024fce91c83068	22	Cooee	40
25ef6c69b2fa60cd024bc2c08a4410caa59f8dad75397a7a7f84e74a825932ac	24	FakeInstall	39
5f87b65bc1c1e07606b1fe3558096d1e1ceb56b262fa57ee328208e2b6ad4470	24	Dowgin	39
7600c278eadfb92ef25145e76132159cc31fc52de7956d28577a96a7b8ad2cec	22	Riskware	39
2e3eb2c85779d85ccbdf6856e22b06fd7eea0d4a3fcd972f6ff0f9605fe453b1	27	Tekwon	38
83e4db70dc9d846a4a0205930196a636c043d6f3c0ee1bfd35bf8148f9134a87	23	Tekwon	38
91d62e582949782f8edf0f7fd685846ca0073be53159d6a2d953344c8c92d2ee	29	Dowgin	38
d17f1f561368ac9f7e46678f7a83590d9dcd5b33c166e1e22a8a1b6e257a2f89	19	Cooee	38

The top 20 Android malware families examined in this study are presented in Table III. The SHA-256, VirusTotal, family, and total permissions are the four columns that compose the table. The secure cryptographic hash function represented by the SHA-256 column creates a distinct 256-bit value for every sample; the values are taken from the VirusTotal platform. The VirusTotal column provides a comparative analysis of malware and benign datasets. Additionally, to supplement the information in the VirusTotal column, the Android malware families found in this investigation are listed in the family column.

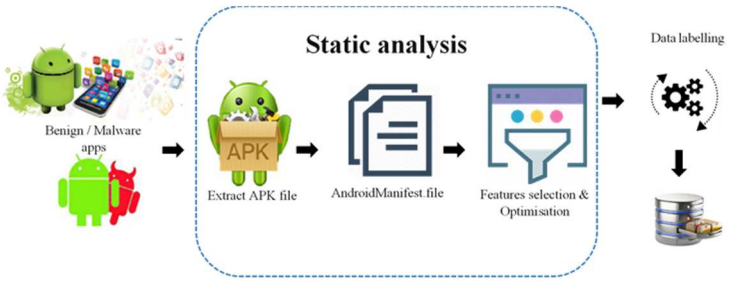

Fig. 2 Dataset collection phase

B. Feature Selection

The feature extraction phase is predominantly centered on permissions, a critical component in the analysis of Android applications. This process involves decompiling the APK file to access the AndroidManifest.xml, which contains permission details, and compiling the classes.dex file, which incorporates Dalvik bytecodes and API calls through static analysis. Advanced tools such as Androguard and ApkTool are employed to extract features from the AndroidManifest.xml and classes.dex files, which collectively provide insights into permissions, system calls, intents, and native code. The AndroidManifest.xml serves as a pivotal resource, encapsulating the application's metadata, including the permissions requested by the application under the <uses-permission/> tag. During this phase, the APK is decompressed, and its binary manifest file is translated into a human-readable XML format. The Androguard tool, in particular, is utilized to precisely extract and analyze permission tags, enabling a thorough investigation of the application's permission structure.

Feature selection, a prevalent pre-processing technique, aims to identify the most relevant attributes by extending and filtering out irrelevant or negatively impactful features on classification outcomes. The benefits include enhancing efficiency and accuracy. While [23] and [24] employed permission-based features and API call sequence features, respectively, [23] and [25] used intent features. The optimization process for permission features is time-intensive during training and testing, but mitigates overfitting, streamlining Android malware detection. Furthermore, optimization can enhance the experiment's accuracy. This process commences with dataset cleansing to eliminate artifacts and superfluous features. Missing data points are filled with zeros before randomization to account for the randomness of the filtering process and eradicate potential biases. The dataset optimization employed the Chi-square algorithm in WEKA, a comprehensive software providing machine learning algorithms [26].

C. Detection phase:

Leveraging deep learning techniques, the proposed system offers a promising approach to enhancing protection against Android malware. Due to their ability to extract relevant, higher-level conceptual features from data, they may provide an effective, broad, and scalable framework for detecting current and unknown Android malware [27]-[31]. Compared to unsupervised learning, a deep learning Multilayer perceptron has a higher detection rate because it is based on supervised learning. The proposed method is described by using two phases: a static analysis phase and a deep learning classifier phase. Figure 3 presents the model of the Android mobile malware detection system, which utilizes deep learning for predictive analysis.

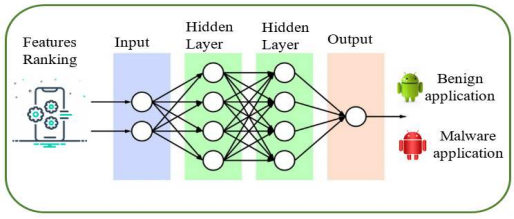

Fig. 3 Deep Learning Malware Detection System

Feature selection is a frequent approach in data pre-processing for machine learning and, to a lesser extent, deep learning. Neural networks inherently function as a black box, often assigning lower weights to features with lesser discriminatory power when generating non-linear combinations. However, this study opted to proceed with feature selection for each of the ten groups containing 96,074 samples. This selection was performed to facilitate a comparison of detection rates with split samples. Chi-square was adopted as the tool for optimizing feature selection. This algorithm was selected for its ability to measure dissimilarity and attain the highest accuracy through feature selection. The study involved the examination of 96,074 Android applications, including 48,820 benign samples and 47,254 malware instances. Feature selection was executed using the Waikato Environment for Knowledge Analysis (WEKA), a sophisticated and adaptable software suite that supports a wide range of machine learning algorithms [8].

To address the binary classification challenge posed by the dataset's binary nature, a multilayer perceptron was developed in this study. During this phase, all required permissions were identified, and features with little potential to distinguish the target class were eliminated. During the learning phase, the dataset was separated into two sections to evaluate the deep learning classifier's detection ability, with 70% training and 30% testing sets. The training set was instrumental in refining the model for precise detection. The dataset was segmented into ten classes, with sample sizes of 10,000, 20,000, 30,000, 40,000, 50,000, 60,000, 70,000, 80,000, 90,000, and 96,074. These samples, encompassing both benign and malware instances, were employed for training. Each iteration was completed within 1 second.

D. Evaluation Measure

The performance of the Android malware detection system was assessed using accuracy and loss metrics. The study employed fundamental metrics to identify Android malware, with accuracy representing the correct identification of malicious content and a higher true positive rate indicating improved performance.

Loss refers to the cost function value during the training phase, while value loss indicates the cost function value during cross-validation, with value accuracy calculated on the validation set.t. Throughout the experiments, various epochs were examined to assess the relationship between accuracy and loss rates. The primary goal was to enhance the detection rate of the deep learning model.

To achieve this, a thorough analysis of the parameters influencing detection performance was carried out. Key parameters, including nodes, epochs, batch size, hidden layers, optimizer, and dropout, were initially set to their default values. These parameters were then incrementally adjusted from their minimum to maximum values to identify optimal configurations. Once the optimal node value was determined, further experiments were conducted to fine-tune the next parameter. To ensure model consistency, random weights, activation functions (RELU and SoftMax), and cross-entropy were applied, but the cost function was maintained. This iterative parameter adjustment process sought to determine the most effective configurations for enhancing malware detection. The experiment encompassed variations in epoch types, batch sizes, and sample sizes. A dataset comprising 96,074 samples was used, which was subsequently divided into ten groups. The experimentation was conducted by increasing the sample sizes. These groups were comprised of 10,000, 20,000, 30,000, 40,000, 50,000, 60,000, 70,000, 80,000, 90,000, and 96,074 samples. To assess result precision and comprehend factors impacting the experiment's accuracy, adjustments were made to both epoch size and batch size. The experiments were conducted under two conditions: one involving feature selection and the other without. This approach aimed to validate the detection capabilities of deep learning with substantial data samples, highlighting its capacity to achieve high accuracy even without feature selection. The experiments were executed with a validation split set at 0.3, a batch size of 500, and four distinct epoch variations. Each epoch iteration was completed in a duration of 1 second.

III. Results and Discussion

The second evaluation discussed in this section was conducted on a dataset comprising 96,074 entries. Within this dataset, there were a total of one hundred sixty (160) permissions considered as features. However, after the optimization procedure, only 20 features were utilized. Section 3 describes the detection framework and discusses the mechanics of the optimization procedure.

A. Evaluation of Accuracy without Feature Selection

Figures 4 to 11 represent the accuracy and loss for two different sample sizes: 10,000 and 20,000 samples. The accuracy of each test validation improved as the number of epochs rose, with accuracy rising and loss decreasing in proportion. Looking at these figures, batch consistency refers to how well a batch of testing data aligns with the training data in terms of accuracy. Loss, on the other hand, represents the degree of error in the data. When accuracy is high and loss is low, it indicates minimal errors in a subset of the data. Loss is calculated during the training process, and its value is determined using the testing data. The validation set was implemented to assess the model's effectiveness, while the training set was used to train the model.

As the number of epochs increased, each test's accuracy improved while the loss reduced. For handling large data samples, a batch size of 500 proved to be the optimal choice, resulting in a reduction of one second in iteration time. We executed 84 iterations, with 30% of the total samples used for validation. Figure 7 depicts the accuracy and loss graphs for 10,000 samples, while Figures 8 to 11 show the corresponding graphs for 20,000 samples. Each graph depicts performance over four (4) separate epochs for each sample size.

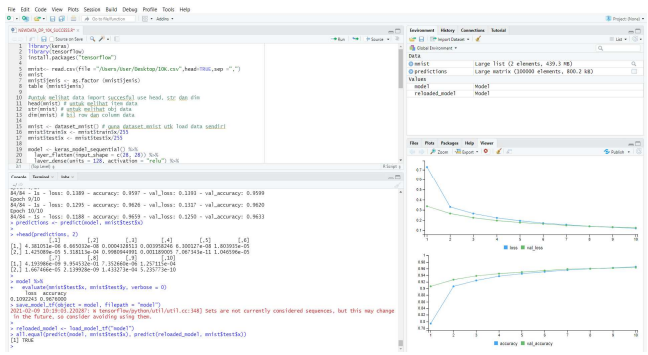

Fig. 4 The 10-EpochsAccuracy and Loss Graph (10,000 Datasets)

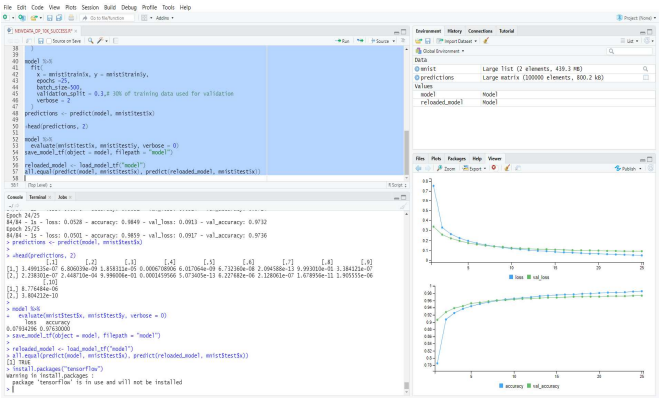

Fig. 5 The 25-Epochs Accuracy and Loss Graph (10,000 Datasets)

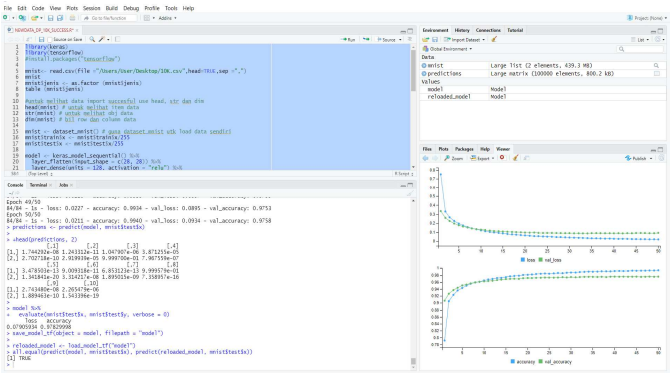

Fig. 6 The 50-Epochs Accuracy and Loss Graph (10,000 Datasets)

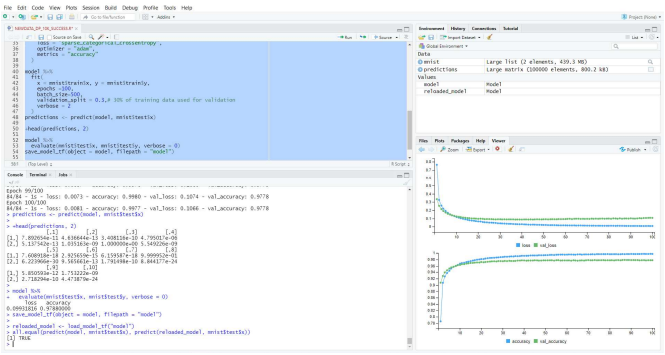

Fig. 7 The 100-EpochsAccuracy and Loss Graph (10,000 Datasets)

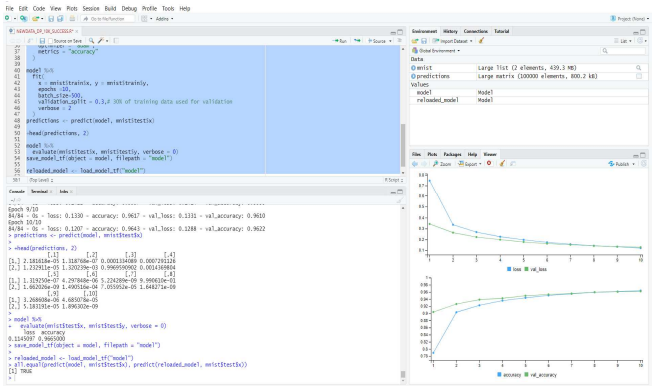

Fig. 8 The 10-Epochs Accuracy and Loss Graph (20,000 Datasets)

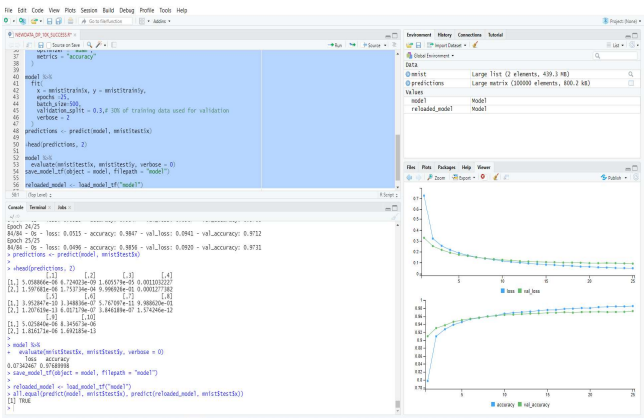

Fig. 9 The 25-Epochs Accuracy and Loss Graph (20,000 Datasets)

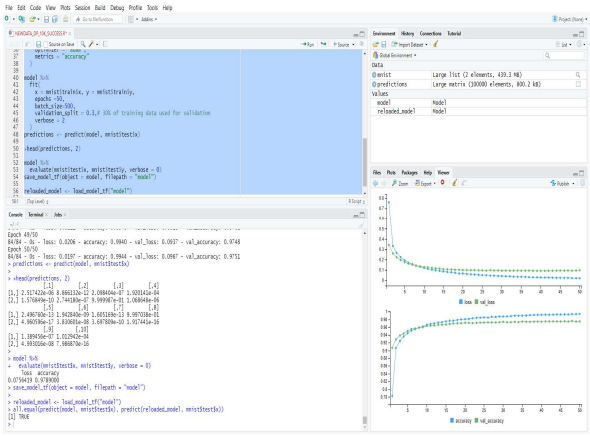

Fig. 10 The 50-Epochs Accuracy and Loss Graph (20,000 Datasets)

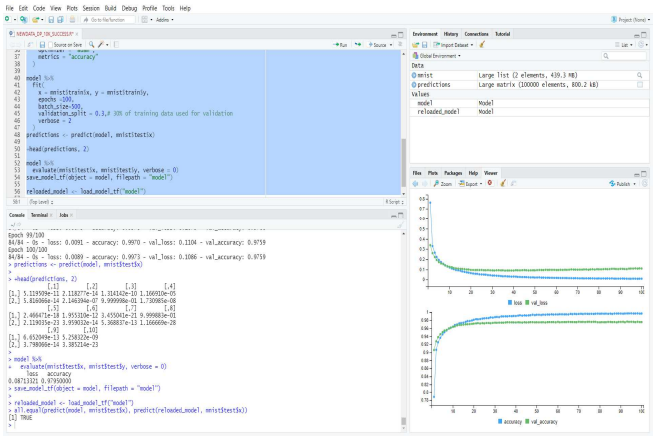

Fig. 11 The 100-Epochs Accuracy and Loss Graph (20,000 Datasets)

TABLE III
ACCURACY OF USING 20 FEATURE SELECTION OF CHI-SQUARE

Samples	Epochs (accuracy)				Epoch (loss)			
	10	25	50	100	10	25	50	100
10,000	0.9673	0.9777	0.9788	0.9802	0.1139	0.0774	0.0775	0.0911
20,000	0.9663	0.9783	0.9786	0.9779	0.1146	0.0744	0.0795	0.0944
30,000	0.9685	0.9773	0.9776	0.9776	0.1091	0.0792	0.0785	0.9500
40,000	0.9685	0.9758	0.9798	0.9787	0.1082	0.0801	0.0738	0.0930
50,000	0.9669	0.9763	0.9771	0.9799	0.1133	0.0800	0.0796	0.0931
60,000	0.9674	0.9764	0.9774	0.9797	0.1122	0.0801	0.0782	0.0914
70,000	0.9672	0.9746	0.9793	0.9784	0.1118	0.0824	0.0713	0.0875
80,000	0.9674	0.9773	0.9770	0.9785	0.1109	0.0775	0.0751	0.0927
90,000	0.9682	0.9773	0.9804	0.9781	0.1088	0.0773	0.0760	0.0979
96,074	0.9660	0.9743	0.9786	0.9784	0.1142	0.0821	0.0775	0.0916

B. Evaluation of Accuracy Using Chi-square Feature Selection

Table III presents the results of malware detection using the deep learning method with Chi-squared feature selection.

Table III displays the accuracy and loss results obtained from the Chi-square algorithm's top 20 feature selection process across different sample sizes and epochs. Increasing the number of epochs led to improved test accuracy and decreased loss. For handling large data samples, the optimal batch size was found to be 500, which added one second to each iteration. A total of 84 batch sizes were produced, with 30% of the total sample set aside for the validation process to complete a single loop. From Figures 12 to 15 show the accuracy and loss trends for 10,000 samples over four distinct epochs, whereas from Figures 16 to 19 show the similar accuracy and loss patterns for 20,000 samples across four different epochs for each sample size.

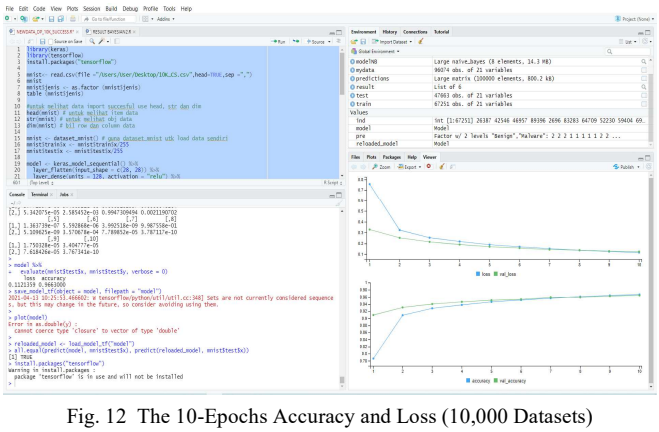

Fig. 12 The 10-Epochs Accuracy and Loss (10,000 Datasets)

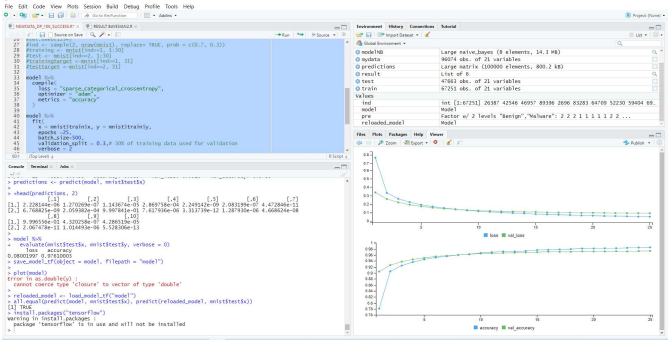

Fig. 13 The 25-Epochs Accuracy and Loss (10,000 Datasets)

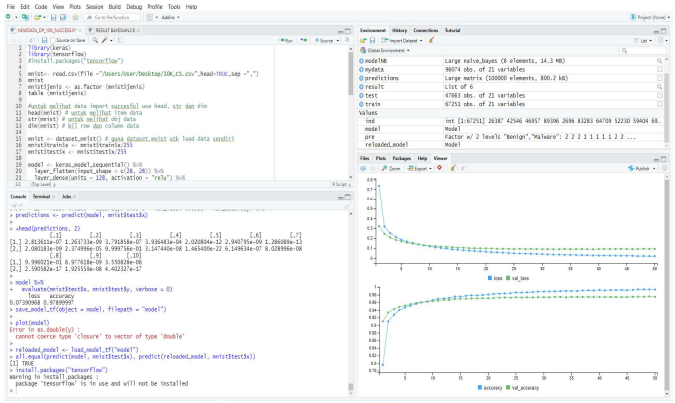

Fig. 14 The 50-Epochs Accuracy and Loss (10,000 Datasets)

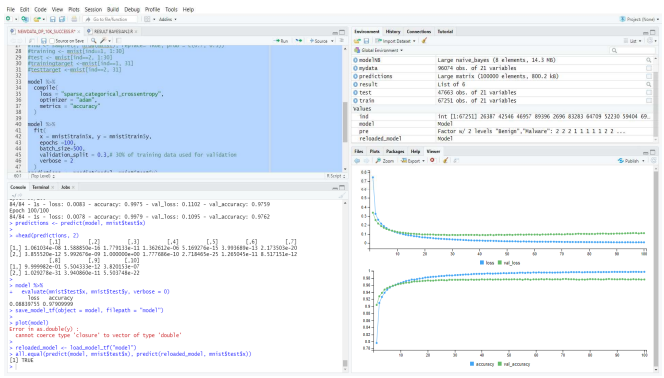

Fig. 15 The 100-Epochs Accuracy and Loss (10,000 Datasets)

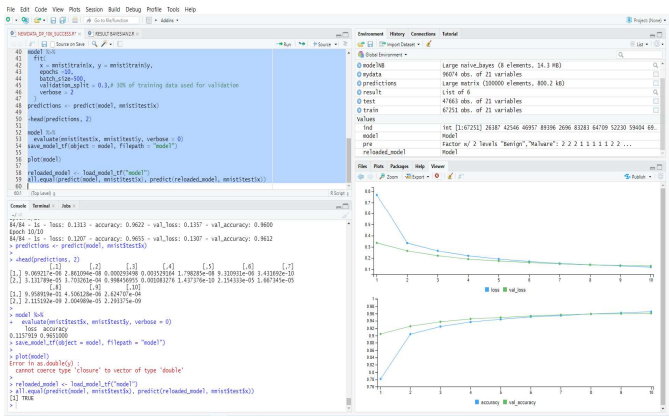

Fig. 16 The 10-Epochs Accuracy and Loss (20,000 Datasets)

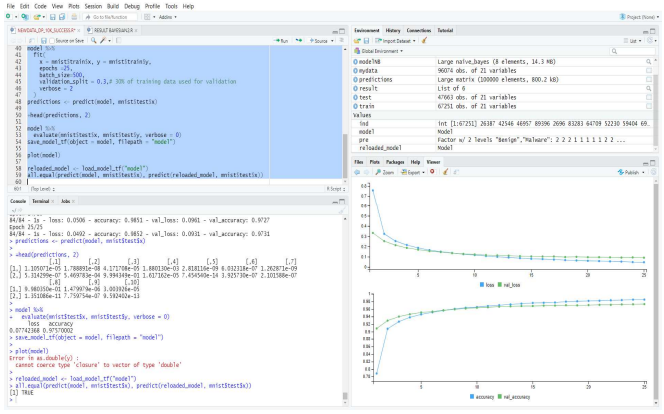

Fig. 17 The 25-Epochs Accuracy and Loss (20,000 Datasets)

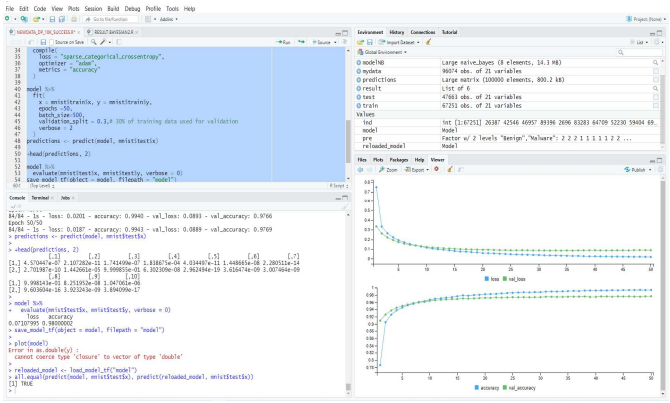

Fig. 18 The 50-Epoch Accuracy and Loss (20,000 Datasets)

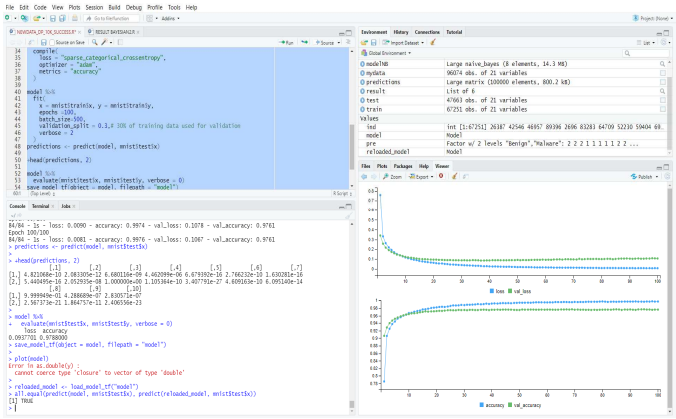

Fig. 19 The 100-Epoch Accuracy and Loss (20,000 Datasets)

The graphs in Figures 12 to 15, as well as Figures 16 to 19, show the accuracy and loss metrics using the Chi-square approach for feature selection, applied to sample sizes of 10,000–20,000, respectively. These results were based on the top 20 permission group features. As the number of epochs increased, test validation accuracy improved, and loss decreased. The consistency of the training data batch, as shown in the figures, reflects the accuracy achieved during testing. Loss represents the error within the data. A higher accuracy with lower loss indicates minimal errors on a relatively small dataset. The loss values for the training and testing datasets were derived. The model was improved using the training set, and its effectiveness and performance were evaluated using the validation set.

IV. Conclusion

The study presented an analysis that utilized a Bayesian classifier to detect mobile malware efficiently. In this work, permission-based features have been selected for malware analysis. The permissions have been extracted from the AndroidManifest.xml file. The best features were identified through optimization using the Information Gain and Chi-Square algorithms. The reason for using two algorithms is to compare the performance and get a higher accuracy rate. Specifically, the study implemented eight experiments, each consisting of 15, 20, 25, and 30 features for both algorithms. The 10,000 samples utilized in the experiments were obtained from Drebin for malware and Androzoo for benign applications. The samples comprised 5,000 benign applications and 5,000 malicious applications. Detection was divided into

three phases: data gathering, pre-processing, and detection. The experiment was implemented with 10-fold cross-validation using WEKA tools. As a result, the Chi-Square algorithm with 15 features outperformed the other features, achieving a 91% accuracy rate. However, there was not much difference in the accuracy rate for the other features in this work. Additionally, analysis of the dataset indicated that the INTERNET permission was the top permission, with 99% of malware and 81% of benign samples. Based on the analysis, the Internet has been used by most people around the world. This study focused on the feature selection of two algorithms in malware detection. Future research could benefit from incorporating risk assessment for each feature, as it is considered a crucial aspect of malware detection. Risk assessment facilitates the classification, prioritization, and zoning of permission requests, helping to identify and mitigate the potential damage posed by malware. This strategy aims to encourage additional research, particularly in overcoming the rapid proliferation of malware. Finally, this paper presents a thorough review of the subject, emphasizing the significance of continued research in Android malware detection.

ACKNOWLEDGMENT

The Ministry of Higher Education FRGS funds the work under Project ID: RDU192613 and RDU192607. The authors thank anonymous reviewers for their constructive comments and University Malaysia Pahang for their support. Communication of this research is made possible through monetary assistance by Universiti Tun Hussein Onn Malaysia and the UTHM Publisher's Office via Publication Fund E15216

REFERENCES

[1] N. S. Nordin and M. A. Ismail, "A hybridization of butterfly optimization algorithm and harmony search for fuzzy modelling in phishing attack detection," *Neural Comput. Appl.*, vol. 35, no. 7, pp. 5501-5512, Mar. 2023, doi: 10.1007/s00521-022-07957-0.

[2] M. F. A. Razak, N. B. Anuar, R. Salleh, and A. Firdaus, "The rise of 'malware': Bibliometric analysis of malware study," *J. Netw. Comput. Appl.*, vol. 75, pp. 58-76, 2016, doi: 10.1016/j.jnca.2016.08.022.

[3] C. Yuan et al., "Android applications categorization using Bayesian classification," in *Proc. Int. Conf. Cyber-Enabled Distrib. Comput. Knowl. Discov. (CyberC)*, 2016, pp. 173-176, doi:10.1109/CyberC.2016.42.

[4] Z. U. Rehman et al., "Machine learning-assisted signature and heuristic-based detection of malwares in Android devices," *Comput. Electr. Eng.*, vol. 69, pp. 828-841, Jul. 2018, doi:10.1016/j.compeleceng.2017.11.028.

[5] O. Koucham, T. Rachidi, and N. Assem, "Host intrusion detection using system call argument-based clustering combined with Bayesian classification," in *Proc. SAI Intell. Syst. Conf. (IntelliSys)*, 2015, pp. 1010-1016, doi: 10.1109/IntelliSys.2015.7361267.

[6] C. H. Liu, Z. J. Zhang, and S. D. Wang, "An android malware detection approach using Bayesian inference," in *Proc. IEEE Int. Conf. Comput. Inf. Technol. (CIT)*, 2016, pp. 476-483, doi: 10.1109/CIT.2016.76.

[7] N. N. M. Nasri et al., "Android malware detection system using machine learning," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 9, no. 1.5, pp. 327-333, 2020, doi: 10.30534/ijatcse/2020/4691.52020.

[8] O. Yildiz and I. A. Doğru, "Permission-based Android malware detection system using feature selection with genetic algorithm," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 29, no. 2, pp. 245-262, 2019, doi:10.1142/S0218194019500116.

[9] H. Hanif et al., "The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches," *J. Netw. Comput. Appl.*, vol. 179, p. 103009, 2021, doi:10.1016/j.jnca.2021.103009.

[10] P. Sreekumari, "Malware detection techniques based on deep learning," in *Proc. IEEE Int. Conf. Big Data Secur. Cloud*, 2020, pp. 65-70, doi: 10.1109/BigDataSecurity-HPSC-IDS49724.2020.00023.

[11] P. Maniriho, A. N. Mahmood, and M. J. M. Chowdhury, "A survey of recent advances in deep learning models for detecting malware in desktop and mobile platforms," *ACM Comput. Surv.*, vol. 56, no. 6, pp. 1-41, 2023, doi: 10.1145/3638240.

[12] P. Saraswathi et al., "An artificial intelligence approach for malware detection using deep learning," in *Data Sci. Big Data Anal.*, 2023, pp. 583-599, doi: 10.1007/978-981-99-9179-2_44.

[13] H. Naeem and A. Batool, "Malware attacks detection in network security using deep learning approaches," *Int. J. Electron. Crime Investig.*, vol. 7, no. 3, pp. 31-44, 2023, doi:10.54692/ijeci.2023.0703160.

[14] V. K. Borate et al., "Analysis of malware detection using various machine learning approach," *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 4, no. 2, pp. 14-321, 2024.

[15] S. Mohan, S. Babu, and B. Sahoo, "Deep learning-based malware detection," in *Proc. Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT)*, 2024, pp. 1-6, doi:10.1109/icccnt61001.2024.10724407.

[16] R. Renugadevi et al., "Malware detection for Android systems using deep learning," in *Proc. Int. Conf. Inventive Syst. Control (ICISC)*, 2024, pp. 67-72, doi: 10.1109/ICISC62624.2024.00018.

[17] O. J. Falana et al., "Mal-Detect: An intelligent visualization approach for malware detection," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 5, pp. 1968-1983, 2022, doi: 10.1016/j.jksuci.2022.02.026.

[18] A. L. Brown, M. Gupta, and M. Abdelsalam, "Automated machine learning for deep learning based malware detection," *Comput. Secur.*, vol. 137, 2024, doi: 10.1016/j.cose.2023.103582.

[19] A. Qamar, A. Karim, and V. Chang, "Mobile malware attacks: Review, taxonomy & future directions," *Future Gener. Comput. Syst.*, vol. 97, pp. 887-909, Aug. 2019, doi: 10.1016/j.future.2019.03.007.

[20] H. Eljak et al., "E-learning-based cloud computing environment: A systematic review, challenges, and opportunities," *IEEE Access*, vol. 12, pp. 7329-7355, 2024, doi: 10.1109/access.2023.3339250.

[21] F. Wu, L. Xiao, and J. Zhu, "Bayesian model updating method based android malware detection for IoT services," in *Proc. Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, 2019, pp. 61-66, doi:10.1109/iwcmc.2019.8766754.

[22] K. Sharma and B. B. Gupta, "Towards privacy risk analysis in android applications using machine learning approaches," *Int. J. E-Services Mobile Appl.*, vol. 11, no. 2, pp. 1-21, Apr. 2019, doi:10.4018/ijesma.2019040101.

[23] A. Feizollah et al., "AndroDialysis: Analysis of Android intent effectiveness in malware detection," *Comput. Secur.*, vol. 65, pp. 121-134, Mar. 2017, doi: 10.1016/j.cose.2016.11.007.

[24] M. A. Jerlin and K. Marimuthu, "A new malware detection system using machine learning techniques for API call sequences," *J. Appl. Secur. Res.*, vol. 13, no. 1, pp. 45-62, Dec. 2017, doi:10.1080/19361610.2018.1387734.

[25] R. Taheri et al., "Similarity-based Android malware detection using Hamming distance of static binary features," *Future Gener. Comput. Syst.*, vol. 105, pp. 230-247, Apr. 2020, doi:10.1016/j.future.2019.11.034.

[26] M. F. A. Razak et al., "Bio-inspired for features optimization and malware detection," *Arab. J. Sci. Eng.*, vol. 43, no. 12, pp. 6963-6979, Nov. 2017, doi: 10.1007/s13369-017-2951-y.

[27] M. Binjubier et al., "A GPU accelerated parallel genetic algorithm for the traveling salesman problem," *J. Soft Comput. Data Min.*, vol. 5, no. 2, pp. 137-150, 2024, doi: 10.30880/jscdm.2024.05.02.010.

[28] A. N. Yussuf et al., "Leveraging deep learning techniques for condition assessment of stormwater pipe network," *J. Civ. Struct. Health Monit.*, vol. 15, no. 1, pp. 619-633, Aug. 2024, doi: 10.1007/s13349-024-00841-6.

[29] J. Purohit and R. Dave, "Leveraging deep learning techniques to obtain efficacious segmentation results," *Arch. Adv. Eng. Sci.*, vol. 1, no. 1, pp. 11-26, Jul. 2023, doi: 10.47852/bonviewAAES32021220.

[30] Aswa, "AI-powered cybersecurity: Leveraging deep learning for real-time threat detection and prevention," *Int. J. Eng. Comput. Sci.*, vol. 14, no. 1, pp. 26758-26772, Jan. 2025.

[31] X. Chen et al., "Leveraging deep learning for automatic literature screening in intelligent bibliometrics," *Int. J. Mach. Learn. Cybern.*, vol. 14, no. 4, pp. 1483-1525, Dec. 2022, doi: 10.1007/s13042-022-01710-8.