Comparison of Single-View and Multi-View Deep Learning for Android Malware Detection

Fika Dwi Rahmawati
Cryptographic Engineering
Poltek Siber dan Sandi Negara
Bogor, Indonesia
fika.dwi@student.poltekssn.ac.id

Raden Budiarto Hadiprakoso
Cryptographic Engineering
Poltek Siber dan Sandi Negara
Bogor, Indonesia
raden.budiarto@poltekssn.ac.id

Ray Novita Yasa
Cryptographic Engineering
Poltek Siber dan Sandi Negara
Bogor, Indonesia
ray.novita@poltekssn.ac.id

2022 International Conference on Information Technology Research and Innovation (ICITRI) | 978-1-6654-6184-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICITRI56423.2022.9970205

***Abstract*— Android is the most rapidly developing smartphone operating system because its open-source nature makes it easier for developers to modify Android applications and functionality. This simplicity has also contributed to a rise in the development of malicious software, sometimes known as malware. Malware typically infects applications to damage the system and steal data, leading to substantial losses for Android users. Therefore, it is essential to take preventative steps to detect malware. Deep learning is one such application method. This study compares single-view and multi-view deep learning architectures to identify Android malware using system calls and permissions. The malware analysis method employed is a hybrid method that combines static and dynamic analysis. Genymotion is used to collect system-call features, whereas Androguard is used to extract permissions. The deep learning base model is created using two unique architectures: LSTM (Long short-term memory) for processing system calls and MLP (Multi-Layer Perceptron) for processing permissions. In single-view deep learning architecture, each feature is treated separately on the model. However, multi-view deep learning features are processed on a concatenated model using the concatenate function. According to the assessment findings, the multi-view deep learning architecture model employing the Adam optimizer and the learning rate parameter of 0.005 achieves an accuracy of 83% and an f1 score of 81%. These findings demonstrate a 2% gain in accuracy over the single view model with the same hyperparameters.**

Keywords—malware analysis, android malware, multi-view deep learning, deep learning

I. INTRODUCTION

Smartphones powered by Android are handy for various tasks, including communication, study, shopping, and entertainment. Android offers benefits over other smartphone operating systems, which explains why the number of Android users continues to rise. Open source benefits the Android operating system, making it the fastest-growing smartphone operating system [1]. It also makes it easy for developers to alter Android operating system applications and functionalities [2]. This convenience may also create malicious software, usually known as malware.

Malware (malicious software) is a computer program or software that can be designed to wreak havoc on a computer system [3]. Malware often infects software or application, wreaking havoc on the Android system and potentially stealing sensitive data from smartphones [2]. As a preventative precaution, Android malware must be analyzed so as not to cause harm to a large number of Android users.

Malware can be analyzed via static, dynamic, or hybrid approaches. Static analysis is a technique for analyzing without executing code. On Android, this method may be implemented via reverse engineering to evaluate the code, for example, by examining the permissions and other manifest file characteristics [4]. The benefit of this approach is that it does not need much time and resources to execute code, but the disadvantage is that it is challenging to examine the obfuscated code [5]. By executing code, the dynamic analysis approach examines malware's interaction with mobile resources and services such as location, network, packets, and operating system. This approach has the benefit of effectively analyzing malware, but it requires a great deal of time and resources. The hybrid analytic technique combines static and dynamic approaches [6]. The adoption of hybrid approaches can improve the detection accuracy of malware [7].

It is common knowledge that a large amount of data is collected from various measurement methods because one type of measurement does not always adequately describe data. For instance, data in images and videos consisting of color and texture information are two different types of data commonly known as multi-view learning. Multi-view learning (MVL) seeks to train one function to represent each view and to combine all views to improve a model's performance [8]. By merging numerous independent features or data sources, MVL seeks to discover shared feature spaces or patterns. With favorable outcomes, the MVL technique has begun to be widely implemented in various domains, including computer vision and sentiment analysis. Although deep learning techniques have begun to be widely used in cybersecurity, other applications connected to MVL have not yet been thoroughly investigated.

In this study, a comparison will be made between single-view deep learning and multi-view deep learning using methodologies from earlier studies. In research [9], a multi-view deep learning architecture was created by integrating multiple categorization models and creating a single output. Meanwhile, a multi-view architecture will be constructed in this study using the classification findings from a single-view architecture. This research will utilize a feature that combines the results of dynamic analysis in a series of system-call calls and the findings of static analysis in the form of permissions. We aim to develop a model for Android malware detection

leveraging multi-view deep learning with Android system-calls and Permissions.

II. Related Works

Numerous studies have been done on malware analysis for the Android platform. Malware and the permission property in the Android manifest file have a significant link, according to Pham et al. [10]. The usage of permission and intent in Android applications, according to Fauzia Idrees [11], is effective and precise in identifying malware while resisting code obfuscation. Intent filter [12], a system command, and fingerprinting [11] are techniques frequently employed in static analysis. For dynamic or behavioral analysis, following system calls [13] and inspecting the instructions [14] are both practical methods.

API calling and opcode sequences in the dynamic analysis are the most efficient, according to a hybrid-based study by Damodaran et al. [15]. However, there is a problem with the ratio of benign samples to malware samples. Therefore, a more thorough analysis is required. Utilizing hybrid-based analysis improves detection, according to several studies. Using static and dynamic analyses, Ali-Gombe et al. examine suspicious behavior and detects resource abuse using static bytecode instrumentation before observing it dynamically [16]. Using conditional dependencies between static and dynamic features (Permission, API calls, and system calls) employed in their machine learning classifiers, Surendran et al. offer a hybrid malware detection mechanism [17]. However, because so many variables need to be studied, this endeavor still has performance problems.

There is a study that compares single-view deep learning vs multi-view deep learning using the Convolutional Neural Network (CNN) architecture to detect Android malware [18]. Opcodes and API calls are utilized for this research. Comparing the two architectures reveals that multi-view deep learning can enhance the performance of the used model.

This research explores further about the user's multi-view deep learning for malware detection. We combined the Android permissions and system-call features. Android Permissions represent static analysis features while system calls represent dynamic features. We implemented the LSTM architecture for system-calls and MLP for analyzing Android permissions.

III. Method

Fig. 1 depicts the research approach followed in this study. In this investigation, single-view and multi-view deep learning systems will be compared. The initial step is preparing the data, which the single-view deep learning model will utilize. The architecture for multi-view deep learning will incorporate the predicted outcomes of single-view deep learning. In addition, each model will be subjected to a model evaluation that will be used to compare single-view and multi-view deep learning architectures.

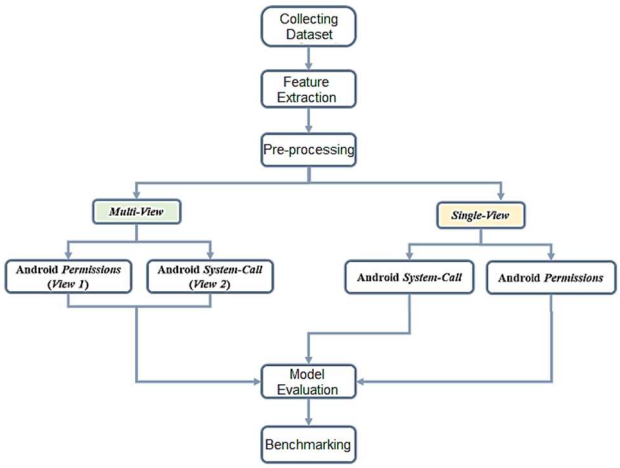

Fig. 1. Research flow.

A. Dataset

The dataset that will be utilized is a comma-separated value (CSV) collection of permissions and system-call calls from Android malware and benignware. The dataset used to train the model is obtained from publicly available sources [19]. The dataset consists of 21992 records, including malicious and benign Android software. 100 Android permissions, 400 system-call sequences, and class labels indicating malware or benignware are included in this collection.

B. Feature Extraction

At this stage, an analysis of the Android application's format is performed. Static analysis and dynamic analysis are employed in the analysis. Permissions utilized by Android applications were extracted from data using static analysis. In the meantime, dynamic analysis is performed by retrieving the sequence of system calls that the Android application makes when it is executed. The system-call extraction procedure employs Genymotion tools and Androguard for permissions extraction. Additionally, the results of the analysis are kept in a text file. In this work, we tested 100 datasets derived from the outcomes of feature extraction.

Data processing permissions are set using the Androguard application on a previously installed virtual host running the Ubuntu operating system. Androguard is a Python-based application that can extract features from Android applications [20]. At this point, Androguard is used to obtain permissions from malware and benignware applications, as illustrated in Fig. 2, which obtained Android permissions from malware applications.

The obtained permissions are then saved in a text file to be processed into a dataset. The stored data is then filtered to remove any irrelevant information. The final step in this process is to transform the data with one-hot-encoding so that deep learning can process it.

System call data processing is performed by tracing system calls on the Android Virtual Device (AVD) running on Genymotion. AVDs running on Genymotion are traced using

Android Debug Bridge (ADB) by looking for the application's process id (PID). Following the acquisition of the application's PID, the next step is to trace the system call that runs on the application and save it to a text file. The obtained system call data cannot be processed directly because it is still raw data containing information not required in the dataset. The next step is to do data pre-processing. At this stage, filtering is carried out to be able to remove information that is not needed by the dataset using the python programming language. During pre-processing, a label-encoding process is also carried out.

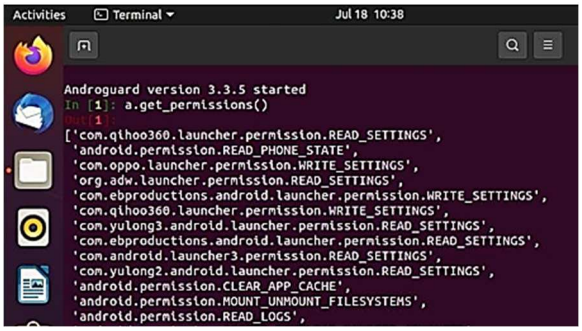

Fig. 2. Android permission feature extraction

C. Modeling

The model will be constructed at this stage by applying single-view and multi-view deep learning for binary classification. Two algorithms, LSTM and MLP, will be utilized. Because the LSTM technique is ideal for sequential data types such as system-call sequences, it will be utilized to process the system-calls feature. In addition, LSTM is more accurate on large data than GRU. As per the study [5], the MLP algorithm will be utilized to process the permissions feature because it is an easy-to-use neural network with good performance for the permissions feature. Using the concatenate function in Keras, the LSTM and MPL models will be integrated in the multi-view deep learning model. Embedding, dropout, dense, optimizer, and bidirectional LSTM are the parameters that will be utilized. At this stage, a search is conducted to determine the ideal parameter values for the optimal model.

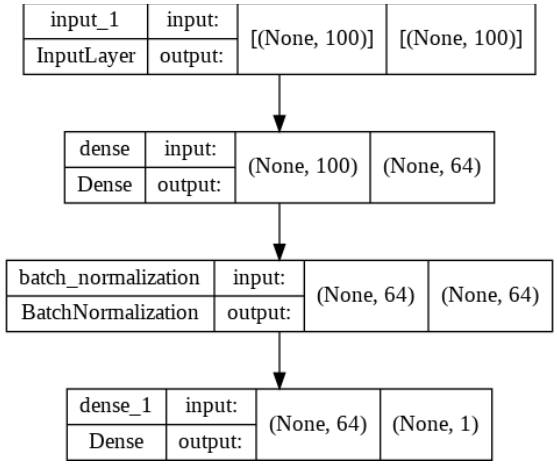

Fig. 3. Single-view Android permission feature model

The Multi-Layer Perceptron (MLP) architecture is implemented for the malware classification process in this study using the Permissions. The result of this model is a model that can predict the class of the malware being tested. The built MLP model consists of input parameters, batch_normalization, and dense. The model diagram can be seen in Fig. 3.

The input dimension in the model is 100 feature Android permissions. These features are entered into the first layer input with 100 neurons. Furthermore, the input variables are entered into the dense layer with 100 input layers and 64 output dimensions for the training process. Then there is a normalization process at the batch_normalization layer to normalize the data again. The output from the Batch Normalization layer is then used as a dense layer to put the output with one neuron with sigmoid activation, which represent the malware label class. The hyper-parameter of MLP model can be seen in TABLE I.

TABLE I. HYPER-PARAMETER OF MLP MODEL

Hyper-parameter	Value
Input	100 Unit
Activation	ReLU, Sigmoid
Dropout	0.5
Optimizer	Adam, RMSProp
Learning Rate	0.02, 0.01, 0.005

We use the LSTM architecture for the system-call based classification model. The built LSTM model consists of input parameters, embedding, Bidirectional LSTM, batch normalization, and dense neuron layers. The result of this model is a class prediction of the malware being tested. The model diagram can be seen in Fig. 4.

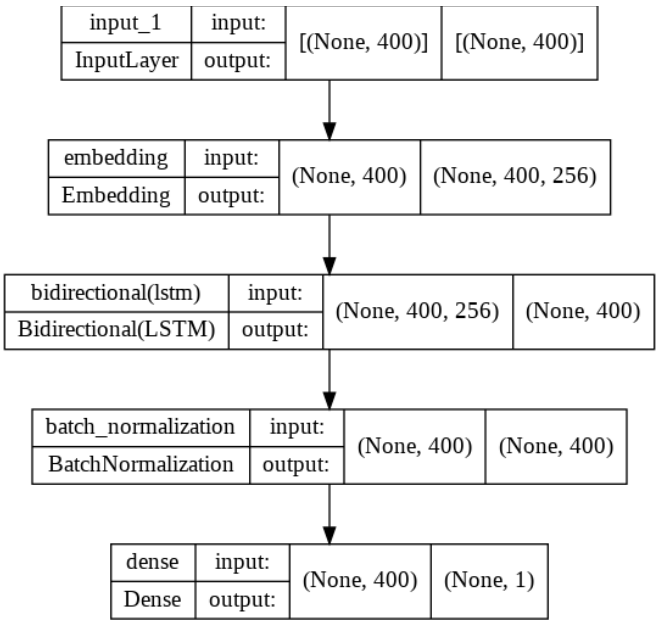

Fig. 4. Single-view Android system-call feature model

The length of the input in the model is 400 system-call sequences consisting of 124 system-call types. The input is

entered into the input layer with 400 input dimensions. Then the input variables enter the embedding layer with 400 input layers and 256 output dimensions. After entering the embedding layer, the input variables enter the LSTM bidirectional layer for further training. Then there is a normalization process at the batch_normalization layer to normalize the data again. The output from the Batch Normalization layer is then used as a dense layer to store output with 1 neuron representing the malware label class. The hyper-parameter of MLP model can be seen in TABLE II.

TABLE II. HYPER-PARAMETER OF LSTM MODEL

Hyper-parameter	Value
Input	400 Unit
Embedding	256 Unit
Activation	Sigmoid
Dropout	0.5
Optimizer	Adam, RMSProp
Learning Rate	0.02, 0.01, dan 0.005

The multi-view model in this study combines the two previous models to predict malware. MLP model to predict permissions and LSTM model to predict system call sequences. The results of the modeling that have been made can be seen in Fig. 5.

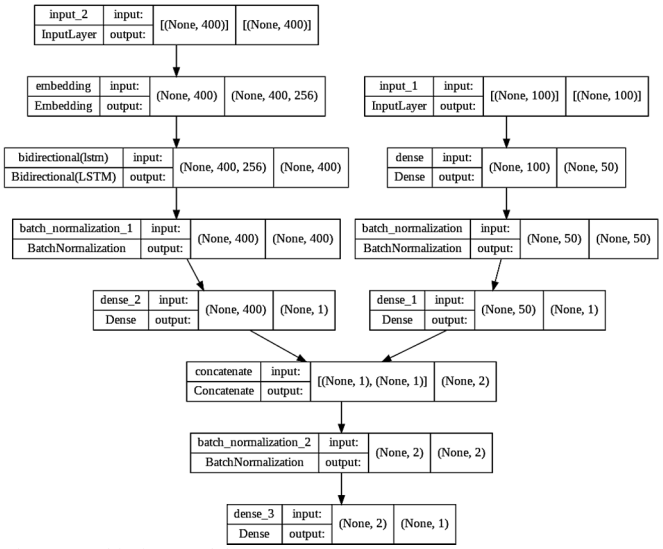

Fig. 5. Multi-view model.

The multi-view model used uses a separate input layer. The LSTM input layer is 400 dimensions, while the MLP layer input is 100 dimensions which describes the number of system calls and permissions used in the dataset. In LSTM, the embedding layer is used for the adjustment process to the input layer with the LSTM layer. The output variables of the two models are combined using a concatenate layer with an output dimension of 2 which describes the output dimensions of MLP and LSTM which is then entered into the batch normalization layer for the data normalization process after concatenating which is then used as output on a dense layer with an output dimension of 1 which describes predictive label variables after training. The hyper-parameter of MLP model can be seen in TABLE III.

TABLE III. HYPER-PARAMETER OF LSTM MODEL

Hyper-parameter	Value
Input	100 Unit, 400 Unit
Embedding	256 Unit
Activation	ReLU, Sigmoid
Dropout	0.5
Optimizer	Adam, RMSProp
Learning Rate	0.02, 0.01, 0.005

D. Model Evaluation

In this phase, the model developed in the previous phase will be evaluated. Accuracy, and the F1-Score will be employed as the measuring metrics. Following this, comparisons are made between single-view and multi-view deep learning architectures to determine the optimal model. Based on the evaluation of the model in the previous phase, benchmarking is performed. Accuracy and f1-score are the measurements utilized at this level. In the benchmarking stage, we perform hyper-parameter adjustments for optimization of the model. The hyper-parameters used in this multi-view model are limited to the Adam optimizer type and the learning rate (0.01, 0.02, 0.05).

E. Environment

Google Colab was utilized throughout this study for data preprocessing, training, and model validation. We also utilize Genymotion as an Android emulator for system call extraction and an open-source tool called Androguard for Android permissions extraction. We use Oracle Virtualbox to build a virtual operating system environment for system-call and permissions extraction. The virtual operating system used is Linux Ubuntu 20.04. The development environment is configured on a Windows 10 system with an Intel Core I7 10700 processor and 16GB of DDR-4 RAM.

IV. RESULT AND DISCUSSION

We utilize Google Colab to process the data after it has been obtained. On the obtained dataset, we received 14 blank and invalid values during the pre-processing data stage. At this moment, all values are removed. When collecting data in a framework-based context, this missing value is primarily caused by corrupt data. The dataset contains 21992 extracted Android applications, 11165 of which are benign and 10827 of which are malicious. Fig. 6 depicts the distribution of the dataset.

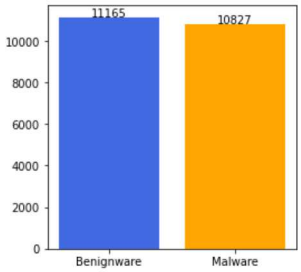

Fig. 6. The distribution of dataset class

A. *Single-View Model*

1) *Permissions based classification model*

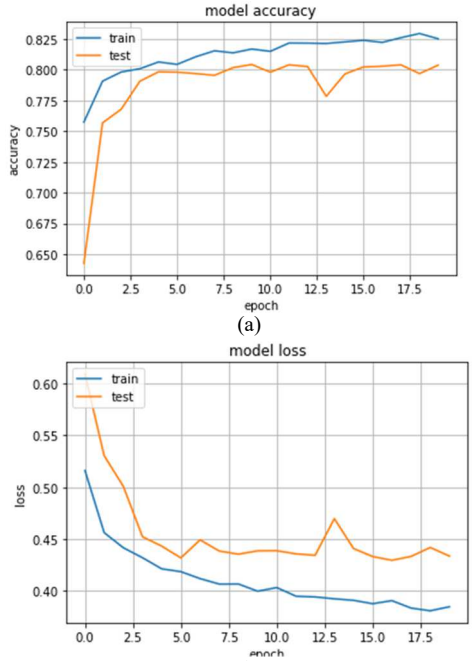

Fig. 7. Plot of MLP model's accuracy and loss on training and testing, (a) Accuracy of training and testing considering Adam and learning rate 0.005, (b) Loss of training and testing considering Adam and learning rate 0.005.

2) *System-call based classification model*

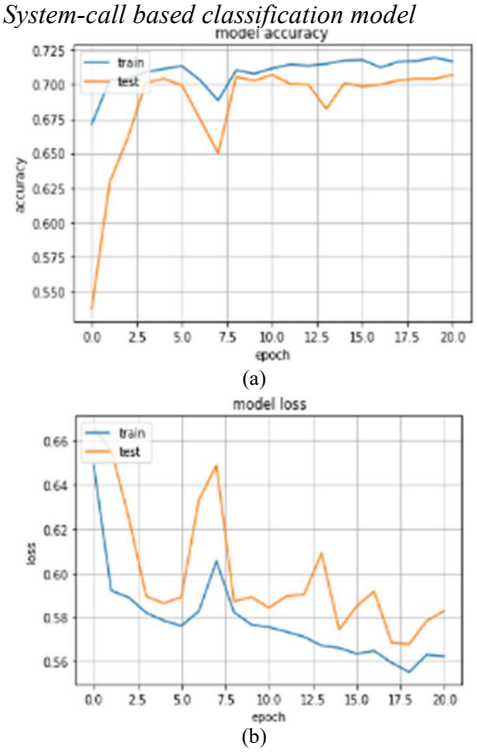

Fig. 8. Plot of LSTM model's accuracy and loss on training and testing, (a) Accuracy of training and testing considering RMSProp and learning rate 0.005, (b) Loss of training and testing considering RMSProp and learning rate 0.005.

Fig. 7 shows the best accuracy and loss of MLP model for both training and testing. The best accuracy and f1-score values are found in the MLP model with Adam optimizer and learning rate value of 0.005. It shows that accuracy and loss of MLP model is fluctuate but each model does not show overfitting and under fitting. It can be seen from Fig. 8 that accuracy and loss of each LSTM model is fluctuate. The best accuracy and f1-score values are found in the LSTM model with RMSProp optimizer and learning rate value of 0.005. However, each model does not show both underfitting and overfitting.

B. *Multi-View Model*

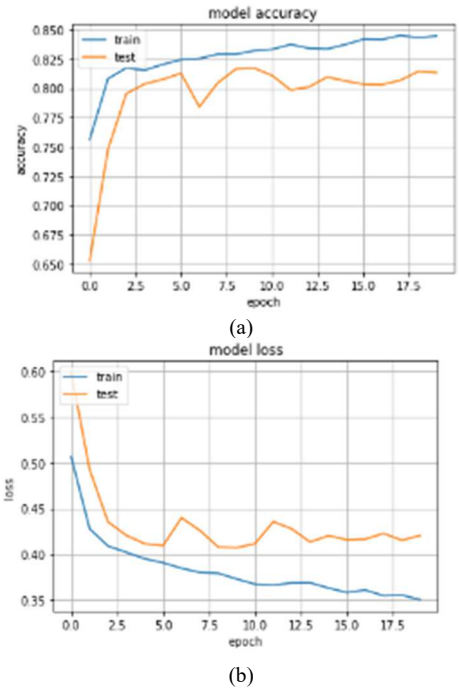

Fig. 9. Plot of LSTM model's accuracy and loss on training and testing, (a) Accuracy of training and testing considering RMSProp and learning rate 0.005, (b) Loss of training and testing considering RMSProp and learning rate 0.005.

TABLE IV. BENCHMARKING OF SINGLE VIEW AND MULTI VIEW DEEP LEARNING

Architecture	Model	Optimizer	Learning rate	Accuracy	F1-Score
Single-view deep learning	MLP	Adam	0.01	0.81	0.79
			0.02	0.80	0.79
			0.005	0.81	0.80
	LSTM	Adam	0.01	0.71	0.67
			0.02	0.70	0.69
			0.05	0.71	0.69
multi-view deep learning	Multi	Adam	0.01	0.82	0.80
			0.02	0.80	0.79
			0.005	0.82	0.81

Plot of accuracy and loss of each multi-view model is shown at fig.9. Both accuracy and loss of each multi-view model is fluctuated but more stable than single-view deep learning. It also does not show both overfitting and underfitting. The best accuracy and f1-score values are found in the multi-view model with RMSProp optimizer and learning rate value of 0.005.

V. CONCLUSION

The following conclusions are drawn from the findings of the comparison research of Single-View Deep Learning and Multi-View Deep Learning for Android Malware detection. According to the model evaluation results, the best model in the single-view deep learning architecture with an accuracy value of 81% and a f1-score of 80% employs the MLP algorithm, and the permissions and optimizer features utilized are Adam with a learning rate parameter of 0.005. While the multi-view deep learning architecture with the Adam optimizer and a learning rate setting of 0.005 achieves 82% accuracy and a f1-score of 81%. As a result, the accuracy value has increased by 1%. Because deep learning uses very large data, the increase in the accuracy value in deep learning makes a big difference even though the increase is only around 1%. The best model for the classification of Android malware, based on the findings of the benchmarking model in this study, combines a multi-view deep learning architecture with the Adam optimizer and a learning rate parameter of 0.005 to achieve an accuracy of 82% and a f1-score of 81%. Because the features used in this study are still limited to Android system calls and permissions, more research is required to investigate other types of features.

REFERENCES

[1] P. S. Uttarwar, R. P. Tidke, D. S. Dandwate, and U. J. Tupe, "A Literature Review on Android-A Mobile Operating system," Int. Res. J. Eng. Technol., no. September, pp. 1–6, 2021, [Online]. Available: www.irjet.net.

[2] S. Herlambang, S. Basuki, D. R. Akbi, and Z. Sari, "Deteksi Malware Android Berdasarkan System Call Menggunakan Algortima Support Vector Machine," vol. 5, pp. 157–165, 2015.

[3] A. P. Namanya, A. Cullen, I. U. Awan, and J. P. Disso, "The World of Malware: An Overview," Proc. - 2018 IEEE 6th Int. Conf. Futur. Internet Things Cloud, FiCloud 2018, no. September, pp. 420–427, 2018, doi: 10.1109/FiCloud.2018.00067.

[4] Q. Wu, X. Zhu, and B. Liu, "A Survey of Android Malware Static Detection Technology Based on Machine Learning," Mob. Inf. Syst., vol. 2021, pp. 1-18, 2021, doi: 10.1155/2021/8896013.

[5] O. S. Jannath Nisha and S. Mary Saira Bhanu, Permission-Based Android Malware Application Detection Using Multi-Layer Perceptron, vol. 941, pp. 362-371. Springer International Publishing, 2020.

[6] M. Ashawa and S. Morris, "Analysis of Android Malware Detection Techniques: A Systematic Review," Int. J. Cyber-Security Digit. Forensics, vol. 8, no. 3, pp. 177–187, 2021, doi: 10.17781/p002605.

[7] R. B. Hadiprakoso, H. Kabetta and I. K. S. Buana, "Hybrid-Based Detection," 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS), 2020, pp. 8-12, doi: 10.1109/ICIMCIS51567.2020.9354315.

[8] J. Zhao, X. Xie, X. Xu, and S. Sun, "Multi-view learning overview: Recent progress and new challenges," Inf. Fusion, vol. 38, pp. 43–54, 2017, doi: 10.1016/j.inffus.2017.02.007.

[9] S. Millar, N. McLaughlin, J. Martinez del Rincon, and P. Miller, "Multi-view deep learning for zero-day Android malware detection," J. Inf. Secur. Appl., vol. 58, p. 102718, January, 2021, doi: 10.1016/j.jisa.2020.102718.

[10] X. Pham, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, "Android malware detection based on system call sequences and LSTM," Multimed. Tools Appl., vol. 78, no. 4, pp. 3979–3999, 2019, doi: 10.1007/s11042-017- 5104-0.

[11] F. Idreesa, M. R. M. Contib, T. M.Chena dan Y. Rahulamathavan, "PIndroid: A novel Android malware detection system using ensemble learning methods," *Computers & Security,* vol. 68, pp. 36-46, 2017

[12] S. Malik, "Android System Call Analysis for Malicious Application Detection," International Journal of Computer Sciences and Engineering, vol. 5, no. 11, pp. 105-108, 2017.

[13] X. Li, J. Liu, Y. Huo, R. Zhang and Y. Yao, "An Android malware detection method based on AndroidManifest file," 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS), 2016, pp. 239-243, doi: 10.1109/CCIS.2016.7790261.

[14] Kabakus, Abdullah Talha, and Ibrahim Alper Dogru. "An In-Depth Analysis of Android Malware Using Hybrid Techniques." Digital Investigation, vol. 24, Mar. 2018, pp. 25–33, 10.1016/j.diin.2018.01.001.

[15] Damodaran, Anusha, et al. "A Comparison of Static, Dynamic, and Hybrid Analysis for Malware Detection." Journal of Computer Virology and Hacking Techniques, vol. 13, no. 1, 29 Dec. 2015, pp. 1–12, 10.1007/s11416-015-0261-z.

[16] Ali-Gombe, Aisha I., et al. "Toward a More Dependable Hybrid Analysis of Android Malware Using Aspect-Oriented Programming." Computers & Security, vol. 73, Mar. 2018, pp. 235–248, 10.1016/j.cose.2017.11.006.

[17] H. Fereidooni, M. Conti, D. Yao and A. Sperduti, "ANASTASIA: Android Malware Detection Using Static Analysis of Applications," 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), 2016, pp. 1-5, doi: 10.1109/NTMS.2016.7792435.

[18] V. Kumar and M. L., "Deep Learning as a Frontier of Machine Learning: A Review," Int. J. Comput. Appl., vol. 182, no. 1, pp. 22–30, 2018, doi: 10.5120/ijca2018917433.

[19] A. de Oliveira and R. J. Sassi, "Chimera: An Android Malware Detection Method Based on Multimodal Deep Learning and Hybrid Analysis," pp. 1– 17, 2020, doi: 10.36227/techrxiv. 13359767.v1.

[20] H. Elfaik and E. H. Nfaoui, "Deep Bidirectional LSTM Network Learning- Based Sentiment Analysis for Arabic Text," J. Intell. Syst., vol. 30, no. 1, pp. 395–412, 2021, doi: 10.1515/jisys-2020-0021.