Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control
Journal homepage: http://kinetik.umm.ac.id
ISSN: 2503-2267
Vol. 9, No. 2, May 2024, Pp.173-180

Optimizing android program malware classification using GridSearchCV optimized random forest

Luqman Hakim*[1], Zamah Sari[1], Ananda Rizaldy Aristyo[1], Syahrul Pangestu[1]
Universitas Muhammadiyah Malang, Indonesia[1]

Article Info
Keywords:
Cyber Security, Malware Classification, Android Smartphones, Random Forest, Hyperparameter Optimization

Article history:
Received: December 29, 2023
Accepted: April 03, 2024
Published: May 31, 2024

Cite:
L. Hakim, Z. Sari, A. R. Aristyo, and S. Pangestu, "Optimzing Android Program Malware Classification Using GridSearchCV Optimized Random Forest", KINETIK, vol. 9, no. 2, May 2024. Retrieved from https://kinetik.umm.ac.id/index.php/kinetik/article/view/1944

*Corresponding author.
Luqman Hakim
E-mail address:
luqman.hakim@umm.ac.id

Abstract
The growing number of smartphones, particularly Android powered ones, has increased public awareness of the security concerns posed by malware and viruses. While machine learning models have been studied for malware prediction in this field, methods for precise identification and classification still require improvement for the perfect detection of malwares and minimizing the cracks on machine learning based classification. Detection accuracy that ranges from 93% to 95% has been observed in prior research, indicates room for improvement. In order to maximize the hyperparameters, this paper suggests improving the Random Forest method by introducing the grid search algorithm which isn't present in previous studies. A significant increase in classification accuracy is the main aim of the research. We exhibit an outstanding 99% accuracy rate in detecting malware contaminated programs, demonstrating the significance of our technique. The proposed method can be seen as a huge improvement over existing models, achieving near perfection in detection, in contrast to which typically obtained by previous models with the accuracy rate of 95% max on the same dataset. Our approach achieves such high accuracy and provides a novel remedy for the limits of the Android based platforms, particularly when program processing resources are limited. This study confirms the effectiveness of our improved Random Forest algorithm, points to a paradigm shift in malware detection, and heightened cybersecurity measures for the rapidly growing smartphone market.

1. Introduction

This increasingly modern era has made almost all of the internet users choose smartphones which are easier and more compact to use when surfing the internet. With the use of smartphones becoming more widespread every year, the targeting of cyber criminal acts will also increase for these platforms. One of the most significant problems facing the internet users today is malware [1]. Malware is a broad term for viruses, worms, trojans, and other malicious software programs that can damage data or access important data illegally [2].

The type of smartphone that is very popular today is a smartphone deployed with the Android operating system [3]. Android as an operating system has the advantages, such as open source code, multitasking capabilities, ease of use, and a very large number of applications supported. The nature of Android as an open source operating system and the broad use of it makes this operating system a promising target in the world of cyber security on the criminal side [4]. Criminal attacks in the cyber domain often occur, especially through the method of spreading malware via the internet [5].

To deal with malware threats, antivirus is now the most commonly used method. On desktop computers, the antivirus will real-time track application data using the latest malware signature files downloaded from the antivirus database. However, implementing the same approach on smartphones is not easy and effective due to significant computational overhead [6]. Using traditional antivirus which only checks the signature and hash value of a file is also increasingly ineffective and considered exhausting in terms of research due to the existence of new malwares that are continuously developing.

The things mentioned above encouraged the authors to propose a more efficient approach for detecting malware threats. With the increasing availability of large datasets and increasingly affordable hardware, the application of deep learning in various factors can become easier to do [7]. Machine learning algorithms seem to offer a solution that can solve the problem of the increasing number of malware, especially using traditional machine learning algorithms. Machine learning algorithms in the form of deep learning neural networks offer more features in their predictions, but are not suitable for application in predicting malware on devices with small resources available, such as smartphones,

because of their large resource requirements [8]. Therefore, in this research, the authors used a traditional machine learning algorithm which is lighter in its development and implementation.

Previously, as seen in Table 1 , a research by Odat et al. (2023) [9] has discussed a model training method that applies a feature coexistence method that allows the model to discover hidden relationships between various features in a dataset [9]. This research produced a high-accuracy model that can predict the DREBIN dataset with an accuracy as high as 95%. Previous research carried out by Arora et al. (2019) [10] applied 'PermPair' which is a detection model with the aim of identifying permission pairs, where if a pair of permissions are combined in a certain pattern, it can indicate the behavior of an application infected with malware. This allows the model to identify potential threats more accurately. A research by Rathore et al. (2021) [11] compared the prediction accuracy of traditional machine learning models with deep neural network models. Neural network is a model that imitates the workings of neural networks in the brain [12]. Their evaluation was also carried out on the same dataset and showed that the traditional random forest classifier machine learning model outperformed other models with an accuracy of 96.9%.

Considering the high level of accuracy obtained by Rathore et al. (2021) [11] using a random forest machine learning model, this research aims to analyze the hyperparameter tuning that can be carried out on the model in order to achieve optimality. We believe that with the right hyperparameter settings, the model can more easily detect hidden relationships between the features available in the dataset. Using these three previous researches [9] [10] [11] as the main references for this research, we aim to perfect the accuracy of the previous researches with the proposed method that would solve the problem that the previous research can't improve anymore: achieving better accuracy within the near-perfect prediction accuracy range and eliminate even the possibility of a tiny bit of imperfection that may occur, potentially providing a security breach if the study were ever to be used in future programs. To optimize the hyperparameters and potentially gain more accuracy, we believe, using grid searching algorithm would help the model in finding the perfect match of hyperparameters combination in no time.

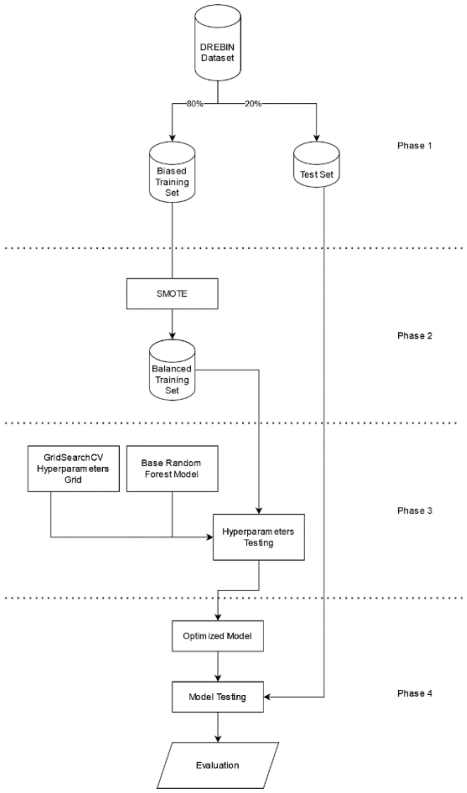

Figure 1. Research Methods

Figure 1 shows the phases and research flow starting from data-splitting, data balancing, determining hyperparameters, model training, to evaluation. The dataset was divided to create two types of dataset, which are the training set and the validation set. This research divides the dataset into two sets with a ratio of 4:1 for the training set and validation set respectively. Dataset cleaning also needs to be done because there is non-uniform data, such as there was a lack of data in several rows. Therefore, empty data must be marked as not available in the dataframe. The

dataset consists of recordings of malware behavior that have been classified, this makes this dataset a dataset resulting from Dynamic Analysis which is an analysis of the behavior of malware [5].

Table 1. Comparison Table of Previous Studies using the Same Dataset

Research	Dataset	Accuracy on The Drebin Dataset	Area of Focus
[9]	DREBIN & MalGenom	95%	The influence of feature coexistence in a dataset on machine learning model predictions
[10]	DREBIN & MalGenom	95%	The influence of the permission pair feature in the dataset on the prediction results of the machine learning model
[11]	DREBIN	96.6%	Comparison of the accuracy of traditional machine learning models with neural network models

2. Research Method

2.1 Dataset

The dataset used to train the model in this research was obtained from the open repository site Figshare [13]. The dataset consists of binary values for the 215 feature categories extracted from 15,036 applications where the application set is divided into 5,560 malware applications from the DREBIN project [14] and 9,476 secure applications. This dataset has a classification imbalance of 13% in the complete dataset where this imbalance can influence and provide bias in determining decisions in future predictions [15]. A model trained on an imbalanced dataset will only be good at predicting the results of the dominant classifications appearing in the dataset, but poor at predicting classifications that are a minority in it. Therefore, data balancing techniques must be implemented to prevent these problems.

The attributes available in this dataset are in the form of API call signatures, which are signatures of various methods or functions available in the Android API. This value represents the action or operation that can be performed by a program [16]. This API call signature determines how the application interacts with various components of the Android system. The next are manifest permissions, which are the permissions declared in the "AndroidManifest.xml" file in an Android application program which determines the access rights needed by the application to perform certain operations or access certain features [17]. This permission regulates the application's security and privacy factors when using the program. Then, the last one is Intent, which refers to the mechanism used for communication between components and applications in the Android operating system [18].

The binary coding of 214 of the 215 available categories indicates the level of Android system access required by an Android application program. The more access permissions given to a program, the more freedom the program has to make changes to the device [17]. In this dataset there is a binary indicator of API access, where the encoding will show a value of 1 if the program calls a particular Android API, and 0 if it does not. Apart from that, this dataset provides an encoding value of 1 or 0 for each condition requiring access based on manifest permission and/or command signature.

2.2 Data Pre-processing

Data splitting is needed to separate the dataset and produce training data and testing data that can be used in training machine learning models and validating models. This time we use training data to test data ratio of 4:1. In other words, 80% of the data is allocated for training the model, and 20% is allocated for testing the model that has been trained. As previously discussed, the dataset used this time has an imbalance in the type of classification results, where the classification of safe programs is more dominant than the classification of programs labeled as malware. Therefore, a data balancing method must be applied.

Oversampling is a technique used to overcome data imbalance problems. One form or method of oversampling that can be used is SMOTE, which is a popular algorithm, and is often used to overcome data imbalances. SMOTE works by creating artificial samples from minority classes through interpolation of previously existing data [19]. This method will balance both sides of the classification as can be seen in Figure 2 and Figure 3 and make the model able to predict both sides of the classification well without any prediction bias.

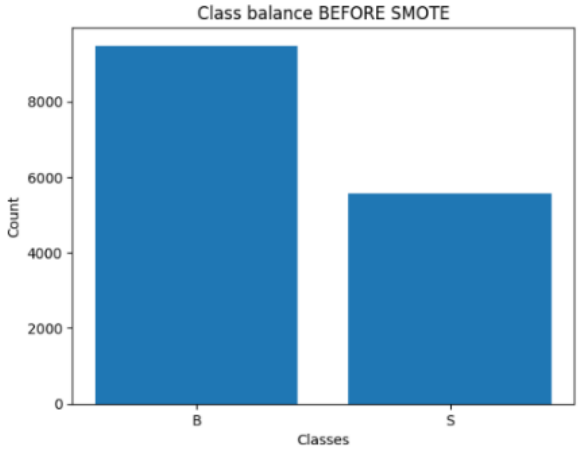

Figure 2. Class Balance Before SMOTE

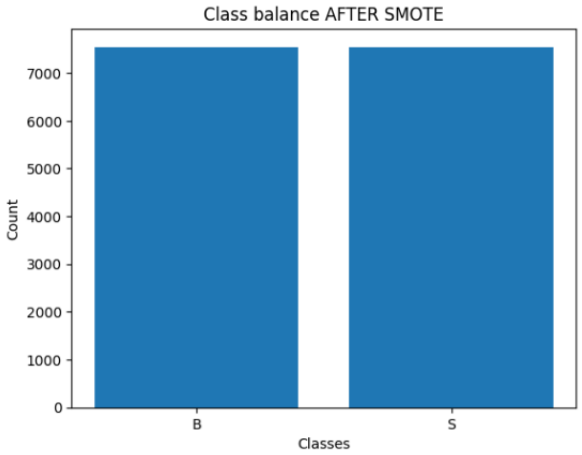

Figure 3. Class Balance After SMOTE

2.3 Machine Learning Model Training Design

As discussed previously, using models that involve neural network methods or other modern methods will only slow down and increase the use of resources on most smartphone hardware. The use of traditional machine learning models will be more profitable in terms of processing power and implementation effectiveness on resource-limited hardware. Therefore, the machine learning model used in this research is a model in the form of a classifier called 'Random Forest Classifier', where this classifier, according to [11], is a model that is relatively light to run and easy to retrain if the data develops at any time. The classifier training method used in this research is integrated with the grid search algorithm, using the 'GridSearchCV' library. This library will carry out training and validation on training data based on all combinations of hyperparameters provided. That way, the model can be trained by looking for the best hyperparameters that can be applied in training [20], this is expected to produce a model that can predict the testing data well. The hyperparameters used in this research are arranged in a grid consisting of 'n_estimators', 'max_depth', 'max_leaf_nodes', and 'criterion'.

Table 2. Hyperparameter Table

Hyperparameter	Value
‘n_estimators’	[100, 200, 300]
‘max_depth’	[None, 5, 10]
‘max_leaf_nodes’	[100, 500, 1000, None]
‘citerion’	[gini, entropy]

‘n_estimators’ determines the number of trees in the random forest, which in the Table 2, there are three values provided. The model will be trained and evaluated with each of these values to determine the best number of estimators for the given data. ‘max_depth’ sets the maximum depth of each decision tree in the model. Higher depth allows the

model to capture more complex correlations in the data, but also increases the risk of overfitting, i.e. a model's great ability at predicting the training set, but poor at predicting the validation set. The value of None in this parameter specifies that the maximum depth is not limited. Next, the 'max_leaf_nodes' value is used to control the maximum number of leaves allowed in each decision tree in the random forest model. This can help the model in controlling the complexity and generalization of the tree. The value of this parameter determines the minimum number of samples required to split the nodes during training. The hyperparameter explanations above are explained and can be found in previous research by Yokoyama et al. (2020) [21]. Lastly 'criterion', this parameter defines the function used to measure the quality of splitting in the random forest. This parameter can be one of the two options 'gini' or 'entropy'. 'gini' is a form of the Gini impurity criterion, while 'entropy' refers to the information gain criterion. Equation 1 and Equation 2 are the formula for measuring Gini index and Entropy impurity respectively [22].

$$Gini = 1 - \sum_{k=1}^{K} P(Y' = k), Y' \subseteq Y \tag{1}$$

$$Entropy = \sum_{k=1}^{K} - P(Y' = k) log_2 . P(Y' = k), Y' \subseteq Y \tag{2}$$

The Gini impurity criterion aims to find features that make up the purest subset of data. A lower Gini impurity index value indicates better class cleavage in the resulting cleavage. Meanwhile, entropy measurements aim to find features that provide a more even distribution of classes between subsets of data. Entropy measures irregularity or randomness in data. The more random or irregular a subset of data is, the higher the entropy value. These criteria help determine the optimal features that can be split at each node.

2.4 Testing Scenarios

In evaluating the performance of the classifier to distinguish between safe and malicious Android applications, we can use formulas and metrics such as: True Positive (TP) which represents the number of correctly identified safe Android applications, True Negative (TN) which represents the number of malicious Android applications correctly identified, False Positive (FP) representing the number of malicious Android applications that were incorrectly identified, and False Negative (FN) representing the number of safe Android applications that were incorrectly identified. From the metrics obtained above, we can calculate the accuracy score of a model's prediction results. The resulting accuracy is a metric that measures the overall correctness of the classifier's predictions. The accuracy score can be calculated using the formula in Equation 3 [22] [23] [24].

$$Accuracy = \frac{tp_i + tn_i}{tp_i + tn_i + fp_i + fn_i} \tag{3}$$

Apart from the accuracy, the precision score (P) can also be taken into account. Precision is a metric that shows the proportion of secure applications correctly identified out of all applications classified as secure. Recall (R) is a metric that shows the proportion of correctly identified secure applications out of all truly secure applications. F1 Score is a metric that combines precision and recall to provide a comprehensive evaluation of classifier performance. Precision, Recall, and F1 Score metrics can be calculated using Equation 4, 5, and 6 respectively [25] [23] [24].

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

$$F1\ Score = \frac{2 * P * R}{P + R} \tag{6}$$

3. Results and Discussion

3.1 Testing Scenarios

After getting the required dataset from [13], as explained in the data pre-processing section, a 4:1 data split will be required, where 80% of the data will be used in model training, while the 20% will be the testing data to evaluate the

results of the model training. The dataset that has been divided certainly has an imbalance between the two classifications. Therefore, it is necessary to apply data balancing using the SMOTE oversampling method.

Table 3. Class Imbalance Contained in the Dataset

Classification	Data Counts
Benign	9476
Suspicious	5560

Table 4. Classification balance after SMOTE implementation

Classification	Data Counts
Benign	7546
Suspicious	7546

Table 3 shows that there is a data difference of 58.69%, which of course will affect the bias of the model's predictions on the testing data and make the model have the potential to experience overfitting and biased prediction. After data balancing, as can be seen in Table 4, the data can be directly consumed by the model. Before the data is consumed by the model, we must find the most optimal hyperparameters to use in this research. Therefore, automated hyperparameter tuning is applied using the GridSearchCV library. As can be seen previously in Table 2, all the hyperparameters are registered in the hyperparameter grid which will later be used as a parameter in the GridSearchCV function.

3.2 Hyperparameter Selection

Next, we will search for the best hyperparameters for the trained model. With the hyperparameter arrangement presented in Table 2, 72 hyperparameter combinations will be produced that can be applied to the random forest model. Meaning that 72 models will be produced from this process, and the best one will be taken as the victor. The grid search algorithm will be very helpful for automatically testing that many parameters because based on the data obtained from this research, doing a grid search with as many hyperparameters as has been mentioned takes approximately 15 minutes on the Google Colaboratory platform, which if done manually, will take longer, hence the potential to increase computing costs.

From the grid search that was carried out previously, the best hyperparameters that were obtained can be applied in this study case.

Table 5. The Best Combination of Hyperparameters

Hyperparameter	Value
‘n_estimators’	200
‘max_depth’	None
‘max_leaf_nodes’	None
‘citerion’	Entropy

Table 5 presents data in the form of hyperparameters and their best values to apply in the model. Where ‘n_estimators’ is assigned a value of 200, ‘max_depth’ has no value, ‘max_leaf_nodes’ also has no value, and ‘criterion’ uses an entropy measurement. Furthermore, data can be drawn from the model with specific hyperparameters, enabling the model to achieve a prediction accuracy of 99%, a perfect score, needless to say. Apart from that, the data that has been taken from the grid search results also stores hyperparameter configurations that have the accuracy following the best hyperparameter combinations.

Table 6. The 9 Highest Hyperparameter Combinations Following the Best

Rank	Hyperparameter	Accuracy
2	{‘criterion’: 'entropy', ‘max_depth’: None, ‘max_leaf_nodes’: None, ‘n_estimators’: 300}	0.9880072331825541
3	{‘criterion’: 'entropy', ‘max_depth’: None, ‘max_leaf_nodes’: 1000, ‘n_estimators’: 300}	0.988007211231891
4	{‘criterion’: 'gini', ‘max_depth’: None, ‘max_leaf_nodes’: 1000, ‘n_estimators’: 200}	0.9880071673305645
5	{‘criterion’: 'gini', ‘max_depth’: None, ‘max_leaf_nodes’: None, ‘n_estimators’: 200}	0.9879410299821915

6	{‘criterion’: 'entropy', ‘max_depth’: None, ‘max_leaf_nodes’: 1000, ‘n_estimators’: 100}	0.9879409641302017
7	{‘criterion’: 'gini', ‘max_depth’: None, ‘max_leaf_nodes’: None, ‘n_estimators’: 300}	0.9879409202288751
8	{‘criterion’: 'entropy', ‘max_depth’: None, ‘max_leaf_nodes’: 1000, ‘n_estimators’: 200}	0.987874695077849
9	{‘criterion’: 'entropy', ‘max_depth’: None, ‘max_leaf_nodes’: None, ‘n_estimators’: 100}	0.9875434815200659
10	{‘criterion’: 'gini', ‘max_depth’: None, ‘max_leaf_nodes’: 1000, ‘n_estimators’: 300}	0.9874109214646974

As presented in Table 6, you can see 9 of the hyperparameter combinations with the highest accuracy following the best combinations. Overall, it can be noticed that the hyperparameter type ‘max_depth’ is strongly biased towards unconstrained values, while the other hyperparameters vary greatly. It seems that limiting the depth of the decision tree in the random forest model in this study will only reduce the accuracy of the model's predictions on test data. This is proven in the combination found at rank 23rd, which starts applying a 'max_depth' value of 10. This combination gets an accuracy score of 97% and consistently decreases to 96% in the combination ranked 47th, where throughout the sequence, everything is identical and uses 10 as the value of 'max_depth'. In other words, there is a degradation of accuracy of 2-3% by limiting the depth of the decision tree to 10 levels of depth. Furthermore, the combination of ratings 48th to 72nd, which all limit the 'max_depth' value to 5, experienced a decrease in accuracy of 6%. Note that the same consistent results can be replicated again by using the 'random_state' parameter with a value of zero. This of course will not affect the results greatly, but it will help the research to be easily validated and replicated accurately in the future studies.

3.3 Feature Importance Similarities From Previous Study

The machine learning model certainly has reasons why a program can be categorized as malware or not. Figure 4 is a by-product of the research, which presents data in the form of a bar chart for 30 features which are the most influential in determining program classification. It can be seen that the highest features are occupied by manifest permission for sending short message services and detecting the status of smartphone information, where this manifest permission is considered dangerous. This is in line with findings in previous research [10].

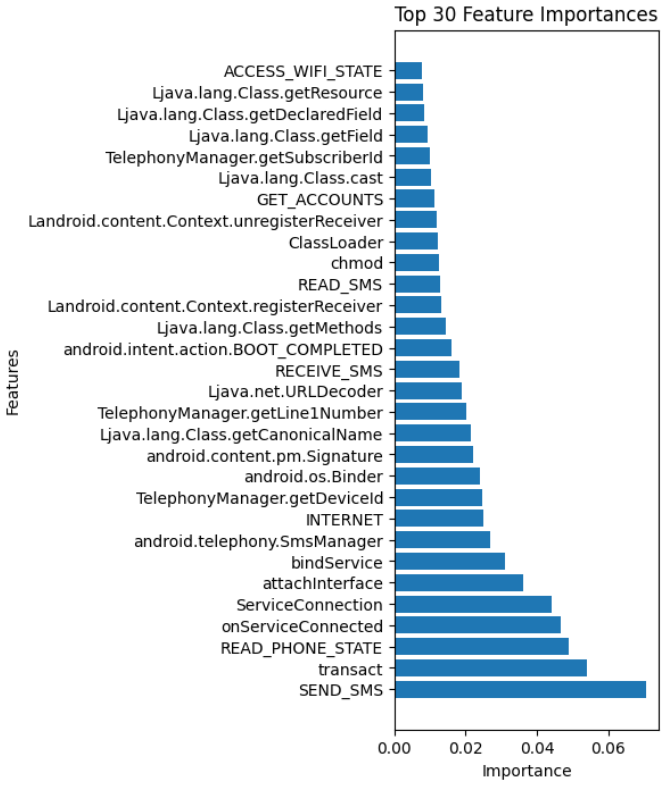

Figure 4. 30 Most Influential Dataset Features According to the Model

4. Conclusion

Optimizing the hyperparameters of the random forest machine learning model by utilizing the grid search algorithm in predicting malware program classification can increase the prediction accuracy on test data by 3% to 6% from the state-of-the-art model. The prediction accuracy is 99%, obtained from applying the best combination of hyperparameters that have been tested using the GridSearchCV library in Python with the provisions of 'criterion' using entropy measurements, 'max_depth' and 'max_leaf_nodes' which are not limited, and 'n_estimators' with a value of 300. It should also be noted that the limitation act of 'max_depth', which is the depth of the decision tree in the random forest model, will only reduce the accuracy of the model's predictions on test data by 2% to 6%. Further research of this method implementation can be broad and variative, it can be developed into pre-market application testing inside a development cycle, or even implemented in an Android endpoint to research the resource used realistically.

References

[1] M. S. Akhtar and T. Feng, "Malware Analysis and Detection Using Machine Learning Algorithms," Symmetry 2022, Vol. 14, Page 2304, vol. 14, no. 11, p. 2304, Nov. 2022. https://doi.org/10.3390/sym14112304

[2] R. Sinha, "STUDY OF MALWARE DETECTION USING MACHINE LEARNING Cyber Crime View project Comparative Analysis of Digital Marketing vs Traditional Marketing View project," University Grant Commision Consortium for Academic and Research Ethics, vol. 51, no. 1, pp. 145–154, 2021. https://doi.org/10.13140/RG.2.2.11478.16963

[3] M. Yang, X. Chen, Y. Luo, and H. Zhang, "An Android Malware Detection Model Based on DT-SVM," Security and Communication Networks, vol. 2020, 2020. https://doi.org/10.1155/2020/8841233

[4] S. K. Sasidharan and C. Thomas, "ProDroid — An Android malware detection framework based on profile hidden Markov model," Pervasive Mob Comput, vol. 72, Apr. 2021. https://doi.org/10.1016/j.pmcj.2021.101336

[5] P. M. Kavitha and B. Muruganantham, "A study on deep learning approaches over malware detection," Proceedings of 2020 IEEE International Conference on Advances and Developments in Electrical and Electronics Engineering, ICADEE 2020, Dec. 2020. https://doi.org/10.1109/ICADEE51157.2020.9368924

[6] M. S. Saleem, J. Mišić, and V. B. Mišić, "Android Malware Detection using Feature Ranking of Permissions," Jan. 2022. https://doi.org/10.48550/arXiv.2201.08468

[7] L. N. Vu and S. Jung, "AdMat: A CNN-on-Matrix Approach to Android Malware Detection and Classification," IEEE Access, vol. 9, pp. 39680–39694, 2021. https://doi.org/10.1109/ACCESS.2021.3063748

[8] I. H. Sarker, "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions," SN Comput Sci, vol. 2, no. 6, pp. 1–20, Nov. 2021. https://doi.org/10.1007/s42979-021-00815-1

[9] E. Odat and Q. M. Yaseen, "A Novel Machine Learning Approach for Android Malware Detection Based on the Co-Existence of Features," IEEE Access, vol. 11, pp. 15471–15484, 2023. https://doi.org/10.1109/ACCESS.2023.3244656

[10] A. Arora, S. K. Peddoju, and M. Conti, "PermPair: Android Malware Detection Using Permission Pairs," IEEE Transactions on Information Forensics and Security, vol. 15, pp. 1968–1982, 2019. https://doi.org/10.1109/TIFS.2019.2950134

[11] H. Rathore, S. K. Sahay, S. Thukral, and M. Sewak, "Detection of Malicious Android Applications: Classical Machine Learning vs. Deep Neural Network Integrated with Clustering," Feb. 2021. https://doi.org/10.48550/arXiv.2103.00637

[12] S. Lu, Q. Li, and X. Zhu, "Stealthy Malware Detection Based on Deep Neural Network," in Journal of Physics: Conference Series, Institute of Physics Publishing, Jan. 2020. https://doi.org/10.1088/1742-6596/1437/1/012123

[13] S. Yerima, "Android malware dataset for machine learning 2." Jun. 2018. https://doi.org/10.6084/m9.figshare.5854653.v1

[14] H. Rafiq, N. Aslam, M. Aleem, B. Issac, and R. H. Randhawa, "AndroMalPack: enhancing the ML-based malware classification by detection and removal of repacked apps for Android systems," Scientific Reports 2022 12:1, vol. 12, no. 1, pp. 1–18, Nov. 2022. https://doi.org/10.1038/s41598-022-23766-w

[15] E. Puyol-Antón et al., "Fairness in Cardiac MR Image Analysis: An Investigation of Bias Due to Data Imbalance in Deep Learning Based Segmentation," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 12903 LNCS, pp. 413–423, 2021. https://doi.org/10.1007/978-3-030-87199-4_39

[16] N. Daoudi, K. Allix, T. F. Bissyandé, and J. Klein, "A Deep Dive Inside DREBIN: An Explorative Analysis beyond Android Malware Detection Scores," ACM Transactions on Privacy and Security, vol. 25, no. 2, May 2022. https://doi.org/10.1145/3503463

[17] K. Khariwal, R. Gupta, J. Singh, and A. Arora, "R MF Droid Android Malware Detection using Ranked Manifest File Components," International Journal of Innovative Technology and Exploring Engineering, vol. 10, no. 7, pp. 55–64, May 2021. https://doi.org/10.35940/ijitee.G8951.0510721

[18] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," in NDSS Symposium 2014, 2014. https://doi.org/10.14722/ndss.2014.23247

[19] D. Dablain, B. Krawczyk, and N. V. Chawla, "DeepSMOTE: Fusing Deep Learning and SMOTE for Imbalanced Data," IEEE Trans Neural Netw Learn Syst, 2022. https://doi.org/10.1109/TNNLS.2021.3136503

[20] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, and Y. Xiang, "A Survey of Android Malware Detection with Deep Neural Models," ACM Computing Surveys, vol. 53, no. 6. Association for Computing Machinery, Feb. 01, 2021. https://doi.org/10.1145/3417978

[21] A. Yokoyama and N. Yamaguchi, "Optimal hyperparameters for random forest to predict leakage current alarm on premises," in E3S Web of Conferences, EDP Sciences, Feb. 2020. https://doi.org/10.1051/e3sconf/202015203003

[22] V. Syrris and D. Geneiatakis, "On machine learning effectiveness for malware detection in Android OS using static analysis data," Journal of Information Security and Applications, vol. 59, p. 102794, Jun. 2021. https://doi.org/10.1016/J.JISA.2021.102794

[23] M. Vakili, M. Ghamsari, and M. Rezaei, "Performance Analysis and Comparison of Machine and Deep Learning Algorithms for IoT Data Classification." arXiv, 2020. https://doi.org/10.48550/ARXIV.2001.09636

[24] M. Grandini, E. Bagli, and G. Visani, "Metrics for Multi-Class Classification: an Overview," Aug. 2020. https://doi.org/10.48550/arXiv.2008.05756

[25] E. S. Alomari et al., "Malware Detection Using Deep Learning and Correlation-Based Feature Selection," Symmetry (Basel), vol. 15, no. 1, Jan. 2023. https://doi.org/10.3390/sym15010123