Android APK Identification using Non Neural Network and Neural Network Classifier

Djarot Hindarto*, Handri Santoso
Magister Technology Informatic, Pradita University
Tangerang, Banten, Indonesia
Email: djarot.hindarto@student.pradita.ac.id, handri.santoso@student.pradita.ac.id

Corresponding Author

Abstract **The purpose of this study is to identify Android APK files by classifying them using Artificial Neural Network (ANN) and Non Neural Network (NNN). The ANN is Multi-Layer Perceptron Classifier (MLPC), while the NNN are KNN, SVM, Decision Tree, Logistic Regression and Naïve Bayes methods. The results show that the performance using NNN has decreasing accuracy when training using larger datasets. The use of the K-Nearest Neighbor algorithm with a dataset of 600 APKs achieves an accuracy of 91.2% and dataset of 14170 APKs achieves an accuracy of 88%. The using of the Support Vector Machine algorithm with the 600 APK dataset has an accuracy of 99.1% and the 14170 APK dataset has an accuracy of 90.5%. The using of the Decision Tree algorithm with the 600 APK dataset has an accuracy of 99.2%, the 14170 APK dataset has an accuracy of 90.8%. The experiment using the Multi-Layer Perceptron Classifier has increasing with the 600 APK dataset reaching 99%, the 7000 APK dataset reaching 100% and the 14170 APK dataset reaching 100%.**

Key words**: Multi Layer Perceptron Classifier, Artificial Neural Network, Non Neural Network, APK Malware Android**

I. INTRODUCTION

Currently, the development of APK malware is increasing, along with the number of Package Kit Applications (APKs) which are applications that run on the Android operating system. So many Androids APKs, causing more and more certain parties to attack for purposes that are profitable for malware makers. Therefore, there are many losses for Android Mobile phones that have been infected by malware. From year to year the development of malware has increased, for this reason this research uses the topic of Android malware. Intents are interfaces that connect interactions between Activities in an Android APK. In addition, Intents send data to other Activities, such as sending data to other applications (Gmail, Google Map, etc.). In essence, Intent is a mechanism to perform an action and communicate between application components.

Originality: Most of the journals in the literature review focus on feature permissions, rarely exploring feature intents. An Android APK to activate an action or activity calls a component, sends data, requires a feature intent. Without feature intents, Android cannot perform action functions. Therefore, this research focuses on feature permissions and feature intents. Malware classification has been carried out by applying machine learning, such as the use of the K-Nearest Neighbor algorithm, Support Vector Machine and Decision Tree. The average classification results are good, but if you use a large dataset, the classification performance decreases. Then the experiment was carried out by applying a deep learning algorithm, namely Multi-Layer Perceptron. Some experimental results continue to increase in accuracy with the increasing number of datasets.

The aim of this study is to identify Android APK files by classifying Android APK files using the Multi-Layer Perceptron Classifier. The main contribution of this paper is to improve the accuracy of malware classification performance by applying the Multi-Layer Perceptron Classifier algorithm.

Some research questions in this study:
RQ 1, How to extract malware dataset using permission feature and intent feature?
RQ 2, What is the percentage of application of the K-NN algorithm, Support Vector Machine and Decision Tree?
RQ 3, What is the percent increase in accuracy with the implementation of the Multi-Layer Perceptron algorithm?
RQ 4, Is it effective to perform malware analysis using static methods?

This article is organized as follows: Section 2 presents related work on several articles related to the Classification of Android malware. Section 3 describes the methodology used. Section 4 describe Propose Method. Section 5 presents the results of the experiments that have been carried out. Section 6 includes a summary of the paper.

II. RELATED WORK

In this study, we compare with previous research that discusses the Android malware APK. The attackers created malware using a new method of targeting victims of Android mobile phones. Several studies have used effective tools to carry out the malware detection process as accurately as possible.

Table I shows a lot of research using extract on feature permissions, system calls, API Calls, Net Info, but still very rarely uses feature intent. This feature intent is an addition to the research, in addition to using feature permissions. This research uses feature permission and feature intent.

TABEL I. LITERATURE REVIEW

Work	Features	Classifier	Data	Performance
[1]	Permission	SVM	10000	Precision 98.20% Recall 95.80% F-measure 96.96%
			25000	Precision 97.16% Recall 93.75% F-measure 95.42%
			60000	Precision 95.17% Recall 92.86% F-measure 94.00%
		DT	10000	Precision 98.99% Recall 96.10% F-measure 97.53%
			25000	Precision 96.10% Recall 93.20% F-measure 94.68%
			60000	Precision 92.11% Recall 91.10% F-measure 91.60%
[2].	Application Programming Interface (API) calls, Permissions	SVM	347 benign, 365 mal	96.2%
		KNN	347 benign, 365 mal	97.2%
		DT	347 benign, 365 mal	96.6%
		RF	347 benign, 365 mal	97.8%
		Naïve Bayes	347 benign, 365 mal	93.9%
		GRU	347 benign, 365 mal	98.2%
[3]	Permission	Chi-Square & Naïve Bayes	5000 mal Drebin and 5000 benign (Andro zo)	91.1%
[4]	API & Permissions	RF ANN	5000 benign, 1260 mal	94% 94%
[5]	Permissions , APIs	SVM	1500 benign, 1500 mal	99.6%
[6]	APIs, Net Info, etc.	SVM K-NN ERT	5560 benign, 5560 mal	90.4% 90.47% 93.66%
[7]	APIs, Net Info	Ensemble	4403 benign, 3982 mal	99.7%
[8]	APIs, Net Info	NB RF K-NN XGBOOST DL	11187 benign, 18677 mal	87% 96% 94% 97% 96%
[9]	Permissions & Intents	BN SVM DT LR	1846 benign, 5560 mal	95.5% 94% 83% 91%
[10]	Permissions & Intents	Ensemble	445 benign, 1246 mal	99.8%

III. METODOLOGY

In this section, the researcher discusses malware analysis and classification [11] research methodology. Performing malware analysis there are three analyzes, namely static analysis[12], dynamic analysis and hybrid analysis. The use of the malware identification or detection method is a supervised learning classification. The algorithm used is KNN, Support Vector Machine and Decision Tree, as well as Deep Learning Multi-Layer Perceptron Classifier[13] [14].

A. Static Analysis

Static analysis [15] is a malware analysis method by analyzing source code. Reverse engineering is used to obtain the source code file, which converts the executable file into a source code file. To analyze the malware APK file, for example, the APK file must be reverse engineered. Analyzing static malware does not need to run the application. Using the JADX module from APKTOOL, to do reverse engineering. The source code to be analyzed is the AndroidManifest.xml file. This file is then read or parse android-permission and android-intent.

Some purposes for reverse engineering:

- To know the protocol of a program. For example: want to create a command line Instagram client.
- To find out the API used by a program. For example, you want to know how to turn on the camera flash as a flashlight.
- To find security bugs for a program.
- To find out if a program violates copyright. For example, we suspect that a program uses a commercial library that we created, without paying for a license.
- For forensic purposes. For example, we want to know the data format used by a program.

E-ISSN:2541-0806
P-ISSN:2540-8895

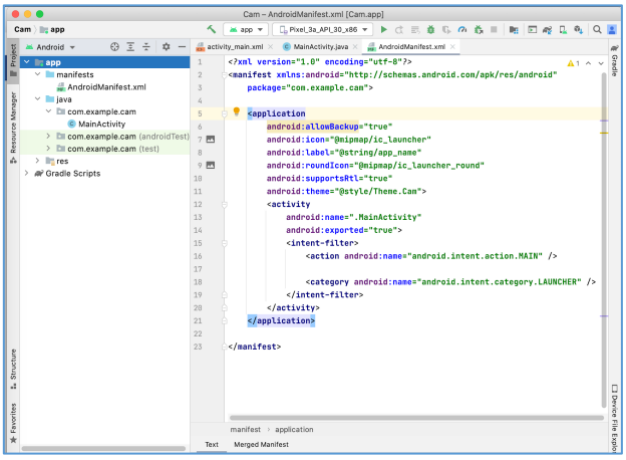

Fig. 1. Architecture Multi-Layer Perceptron

Figure 1, AndroidManifest.xml file, the result of the reverse engineering process. This file will parse the permissions and intent features.

B. Dynamic Analysis

Malware is a threat to Android, various methods are used to analyze malware, one of which is using dynamic analysis. Analyzing Android malware with dynamic methods aims to understand its behavior and improve the ability to detect it. Dynamic analysis also takes an analytical approach to analyze Android malware behavior. How to perform analysis by running malware code in a virtual environment to understand the actual behavior of malware. The dynamic analysis method does not examine the source code, but runs the malware files in a controlled environment, which is called a sandbox. This way the behavior of the malware can be analyzed in a controlled environment, this is very useful where the malware does not spread to other systems. After observing the behavior of malware, a log of malware activity is obtained. This log will be analyzed.

C. Hybrid Analysis

Dynamic malware analysis is a combination of static analysis and dynamic analysis, where the analysis runs the malware in a controlled environment after that it also analyzes the source code. Hybrid model analysis is a perfect and complete analysis for analyzing a malware.

D. K-Nearest Neighbor

K-Nearest Neighbor (KNN) [16] [17] is a classification algorithm using a way to measure the distance, which is measured from the k nearest neighbors. This classification projects the training dataset in a multidimensional space. The space is divided into sections that describe the character of the data. Each training data is represented as points in a multidimensional space. Where the KNN classification[18] [19] process is looking for the point c closest to the new (c). The general formula is to find the Euclidean distance, Hamming distance, Manhattan distance, and Minkowski Distance.

Euclidean distance [20] is a formula for finding the distance between two points in two-dimensional space. Hamming distance [21] is a way to find the distance between two points which is calculated by the length of the binary vector formed by the two points in the binary code block. Manhattan Distance [22] is a formula to find the distance d between 2 vectors in n dimensional space. Minkowski distance is a formula for measuring between two points in a normal vector space which is a hybridization that generalizes the Euclidean distance and Manhattan distance. The K-Nearest Neighbor (KNN) [23] [24] algorithm is a classification of objects based on the learning data that is closest to the object. Then the determination of the K value is carried out. It is determined that the K value is odd, after that a vote is carried out on the closest distance. Advantages of KNN (K-Nearest Neighbor), dataset used for training is very nonlinear and easy to implementation. Disadvantages of KNN (K-Nearest Neighbor): Need to indicate the parameter K (number of nearest neighbors). Does not handle missing values implicitly. Sensitive to data outliers (outliers). Vulnerable to non-informative variables. Vulnerable to high dimensionality. The computational cost is quite high, because it is necessary to calculate the distance from each testing data to the entire training data.

E. Support Vector Machine

Support Vector Machine (SVM) was first presented by Boser, Guyon and Vapnik in 1992. Support Vector Machine is a supported classification algorithm by finding the hyperplane with the largest margin. There are three main sections in SVM, namely Supervised, Classified and Hyperplane with the largest margin. How the Support Vector Machine [25] [26] [27] works. Support vectors are two closely spaced data that come from different classes or groups, these two data will be used as support vectors. Hyperplane [28] [29] is the dividing line between support vectors. Max Margin[30] is the distance between the support vector and the hyperplane, the margin distance must be maximum to be able to anticipate the similarity of one data to another. For non-linear data, SVM Kernel Trick [31] [32] [33] is used by creating new dimensions. So that it can create a hyperplane. The advantage of SVM is that Supervised is able to control the accuracy of classification and Kernel trick is able to classify with non-linear data. Disadvantages of SVM, not good for large amounts of data and Kernel trick is not easy to implement.

F. Decision Tree

The Decision Tree [34] [35] algorithm was developed by J. Ross Quinlan, in 1975. Decision tree is a popular classification method, because it is easy to interpret. Predictive model that uses a tree structure. Another term for Decision Tree is Classification and Regression Tree (CART)[35] [36] [37] which is a decision tree. Decision trees can convert data into decision trees and decision rules. The benefits of DT are its ability to break down complex decision-making processes into simpler ones, so that decision-makers better interpret problem solutions.

Making a Decision Tree model[38] [39] is like drawing an inverted tree where the Root Node is in the top position. Internal Node that has 1 input and at least 2 outputs. Leaf Node is the final Node, has 1 input and has no output.

G. Multi-Layer Perceptron Classifier

Multi-Layer Perceptron is a classification algorithm that works by using a deep neural network. This algorithm is very different from machine learning algorithms based on statistical science. By using the deep neural network method, it is expected that the performance of the model is more accurate, when compared to machine learning. Figure 2 is the architecture of the Multi-Layer Perceptron Classifier to complete the classification of the malware dataset.

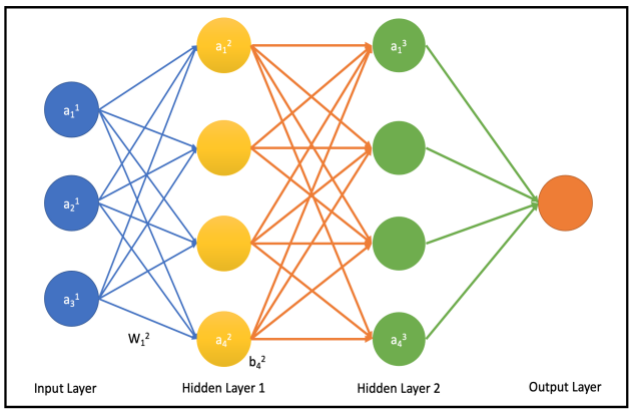

Fig. 2. Architecture Multi-Layer Perceptron

IV. Proposed Method

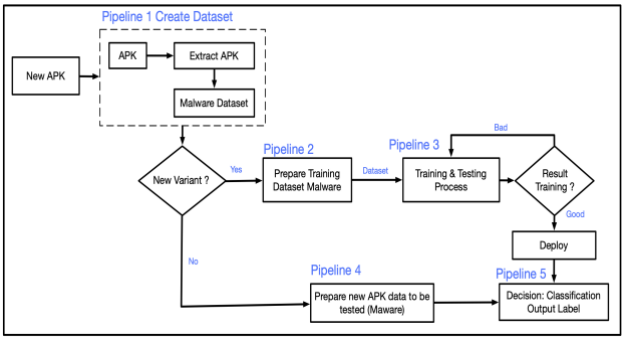

Fig. 3. Architecture Multi-Layer Perceptron

A. Pipeline 1: Create Dataset

This stage is to create a dataset from Android APK files that are indicated as malware or Benign. The malware APK files are downloaded from the University of New Brunswick. The file has been labeled for types of malwares. The downloaded file is accommodated to local storage, then the classification process is carried out and stored in a similar folder. Next, the Android APK file extraction feature is carried out using reverse engineering. Many reverse engineering tools are commonly used. In this research, reverse engineering uses the JADX module. The result of the reverse engineering process is some folders and files AndroidManifest.xml. Files and folders other than AndroidManifest.xml are deleted, while AndroidManifest.xml is then parsed to read the permissions and intent features. The results of the feature extraction process produce a malware dataset. The next process is classification using machine learning or deep learning algorithms.

B. Pipeline 2: Prepare Training Dataset malware

Before training the malware dataset, the prepare stage is very necessary. To generate a model from a machine learning or deep learning training process must use a clean dataset, a good dataset (no null, incorrect data in features). The dataset must ensure that the contents of the malware Dataset should not be mixed with the Benign data. If there is a mixture of malware and Benign, the resulting model will experience errors and affect the performance of the model. In addition to the data cleaning process, there are also engineering features, namely feature analysis and the most influential features. This process must be carried out because this process is also very influential on the resulting model. The next process is to create a uniform dataset, in the sense that if there are five groups of datasets, then the dataset must be an unmixed dataset. For example, the malware APK dataset is of the Ransomware type, then the Ransomware dataset should not be mixed with the Riskware APK dataset.

The division of the number of datasets for machine learning is to divide the 70% training dataset and 30% testing data. But there is no requirement to do so. There are also those who share it, 60% training data and 40% testing data. Sharing datasets for deep learning, training data, validation data and testing data. Example (Data Training + Data Validation) = 70%, while testing data is 30%. Cross validation of datasets or swapping training positions with testing is also carried out to get the performance model that will be generated by machine learning or deep learning.

Some of the reasons for this data preparation is done:

1) *The data owned is not ideal, there is data that is missing value*. Missing data in the dataset will result in a declining model for its performance. Filling must be done so that the dataset becomes intact and good. It is not permissible to fill in the dataset arbitrarily and an analysis of the features or dimensions of the appropriate dataset must be carried out.

2) There are different data formats. To avoid differences in formats in the feature dataset, it is necessary to check, validate the dataset and analyze the features of the dataset.

3) Small dataset or imbalanced dataset from ideal in terms of quantity. Small datasets are not ideal for machine learning or deep learning processes to be generated as models. This makes the model invalid. Synthetic Minority Over-sampling Technique (SMOTE) is a way to balance the dataset, if machine learning is done, in order to produce a good model.

4) The dependent variable and the independent variable are not clear or have no label.

C. Pipeline 3: Training and Testing Process

This stage is conducting training on the malware dataset. Training using the KNN Algorithm, Support Vector Machine and Decision Tree. The distribution of the dataset is carried out, the training dataset is 70% and the testing dataset is 30%. The Multi-Layer Perceptron Classifier algorithm is also used for this stage. The training process is also carried out using changes in the position of the training dataset and testing dataset, which is better known as cross validation. In this study using 5-fold cross validation, to get better model accuracy.

Cross Validation (CV) is a method used to evaluate model performance, where data is separated into two subsets, namely learning process data and evaluation data. The model or algorithm is trained by the learning subset and validated by the validation subset. Furthermore, the selection of the type of CV can be based on the size of the dataset. CV K-fold is used because it can reduce computation time while maintaining the accuracy of the estimate. 5-fold CV is one of the K-fold CVs used for selecting the best model because it tends to provide less biased accuracy estimates. In 5-fold CV, the dataset is divided into 5 folds of approximately equal size, thus having 5 subsets of data to evaluate model performance. For each of these 5 subsets of data, CV will use 4 folds for training data and 1-fold for testing.

D. Pipeline 4: Prepare New APK data to be tested

At this stage the aim is to add new datasets. If in performing the classification and new variants of malware are found, before being entered into the dataset, the data must be feature extraction. Then retraining is carried out. The more datasets, the better the classification model in identifying malware APK.

E. Pipeline 5: Decision Classification Output Label

The last stage aims to produce a classification model and the model is ready for deployment. Testing the model before the model is ready for use, aims to anticipate model errors in identifying Android APK files.

V. Experiment and Result

In conducting the experiment, using the MacBook Air 2020 hardware with specifications of 8 GB RAM, 256 GB storage. Using the Python programming language in the Jupiter Notebook package, the reverse engineer JADX module made by APKTOOL. In this section, answer research questions and report experimental results.

A. RQ 1, How to extract malware dataset using permission feature and intent feature?

This is a much-needed step, where this step generates a malware dataset. APK files are downloaded and extracted, reverse engineered and parsed to read feature permissions and feature intents. The final result of feature extraction is a malware dataset. Following are the feature-feature permissions of the malware dataset:

TABLE I. Normal Feature Permission

ACCESS_ALL_DOWNLOADS	ACCESS_CACHE_FILESYSTEM
BLUETOOTH_SHARE	ACCESS_CHECKIN_PROPERTIES
ACCESS_DOWNLOAD_MANAGER_ADVANCED	ACCESS_DRM_CERTIFICATES
ACCESS_NETWORK_CONDITIONS	ACCESS_NOTIFICATIONS
ACCESS_NETWORK_STATE	ACCESS_NOTIFICATION_POLICY
ACCESS_PDB_STATE	BLUETOOTH_PRIVILEGED
BIND_DEVICE_ADMIN	BIND_CARRIER_SERVICES
BIND_CARRIER_MESSAGING_SERVICE	BIND_APPWIDGET
BIND_ACCESSIBILITY_SERVICE	BROADCAST_PHONE_ACCOUNT_REGISTRATION
BROADCAST_SMS	BIND_ACCESSIBILITY_SERVICE
CAPTURE_AUDIO_OUTPUT	CALL_PRIVILEGED
CONFIGURE_WIFI_DISPLAY	CHANGE_CONFIGURATION
INSTALL_GRANT_RUNTIME_PERMISSIONS	INTERNAL_SYSTEM_WINDOW
INSTALL_LOCATION_PROVIDER	INTERACT_ACROSS_USERS_FULL
GET_PASSWORD	KILL_BACKGROUND_PROCESSES
INTERNAL_SYSTEM_WINDOW	INSTALL_GRANT_RUNTIME_PERMISSIONS
MANAGE_DOCUMENTS	MANAGE_ACCOUNTS
HARDWARE_TEST	INSTALL_LOCATION_PROVIDER
FORCE_STOP_PACKAGES	MANAGE_APP_TOKENS
INSTALL_PACKAGES	KILL_UID

TABLE II. Normal Feature Intent

ACTION_POWER_CONNECTED	ACTION_POWER_DISCONNECTED
ACTION_SHUTDOWN	AIRPLANE_MODE
BATTERY_CHANGED	BATTERY_LOW
BATTERY_OKAY	BOOT_COMPLETED
CAMERA_BUTTON	CONFIGURATION_CHANGED
CREATE_SHORTCUT	DATE_CHANGED
DEVICE_STORAGE_LOW	DEVICE_STORAGE_OK
DOCK_EVENT	DREAMING_STARTED

DREAMING_STOPPED	EXTERNAL_APPLICATIONS_AVAILABLE
EXTERNAL_APPLICATIONS_UNAVAILABLE	FETCH_VOICEMAIL
GTALK_CONNECTED	GTALK_DISCONNECTED
HEADSET_PLUG	INPUT_METHOD_CHANGED
LOCALE_CHANGED	MANAGE_PACKAGE_STORAGE
MAIN	MEDIA_BAD_REMOVAL
MEDIA_BUTTON	MEDIA_CHECKING
MEDIA_MOUNTED	MEDIA_REMOVED
PACKAGE_ADDED	PACKAGE_DATA_CLEARED
PACKAGE_CHANGED	MEDIA_SCANNER_SCAN_FILE
PACKAGE_INSTALL	PACKAGE_NEEDS_VERIFICATION
PACKAGE_REMOVED	PROVIDER_CHANGED
ACTION_TIME_CHANGED	SIM_STATE_CHANGED
SENT_SMS_ACTION	ACTION_EXTERNAL_APPLICATIONS_AVAILABLE

Furthermore, the permission features and intent features are trained with machine learning and deep learning to produce models with the best accuracy. In performing the extraction of the Android APK dataset consisting of the Benign APK and the malware APK that have been labeled, it takes 24 hours of processing for 2 weeks. Table II and Table III are features generated by the feature extraction process from malware and benign android APK files. The total of downloaded APK files = 14,170 APKs. Where the APK file process is reverse engineered with the Jadx APKTOOL module. The results of reverse engineering produce the source code of the APK. Next take the AndroidManifest.xml file to parse the permission features and intent features. Feature parse results are stored into the malware dataset. The malware dataset has permission and intent features of 1178 columns or dimensions. Table II shows some of the permission features and Table III shows some of the intent features.

B. RQ 2, What is the percentage of application of the K-Nearest Neighbor algorithm, Support Vector Machine algorithm and Decision Tree algorithm?

Definition
TP = True Positive.
TN = True Negative.
FP = False Positive.
FN = False Negative.

Accuracy is the ratio of correct predictions (positive and negative) to the entire dataset. Accuracy and answer the question “What percentage of Android APK files correctly predicted Malware and Benign from the entire dataset of Android APK files”. Accuracy = (TP + TN) / (TP + FP + FN + TN). Accuracy can be seen in table IV.

TABLE III. ACCURACY FOR ALGORITHM MACHINE LEARNING

Algorithm	Accuracy		
	600 APK	7000 APK	14170 APK
K Nearest Neighbour (KNN)	88%	86.8%	88%
Decision Tree (DT)	100%	89%	91,3%
Support Vector Machine (SVM)	97%	90%	91%

Precision is the ratio of a positive correct prediction compared to the overall positive predicted outcome. Precision answers the question “What percentage of Android APK files are Malware correct from the total dataset that Malware predicts?”. Precision = (TP) / (TP + FP). Precision can be seen in Table V.

TABLE IV. PRESSION FOR ALGORITHM MACHINE LEARNING

Algorithm	Precision		
	600 APK	7000 APK	14170 APK
K-Nearest Neighbour (KNN)	88%	85.6%	88%
Decision Tree (DT)	100%	89.4%	91.8%
Support Vector Machine (SVM)	96.5%	90%	91.4%

F1 Score is a weighted comparison of the average precision and recall. F1 Score = 2 * (Recall * Precision) / (Recall + Precision). F1-Score can be seen in Table VI.

TABLE V. F1-SCORE FOR ALGORITHM MACHINE LEARNING

Algorithm	F1-Score		
	600 APK	7000 APK	14170 APK
K-Nearest Neighbour (KNN)	88%	85.2%	88%
Decision Tree (DT)	100%	88.6%	91.2%
Support Vector Machine (SVM)	96.7%	89.4%	90.4%

Recall is the ratio of true positive predictions compared to the total number of true positive data. Recall answers the question "What percentage of Android APK files are predicted to be malware compared to all students who are actually malware". Recall = (TP) / (TP + FN). Recall can be seen in Table VII.

TABLE VI. RECALL FOR ALGORITHM MACHINE LEARNING

Algorithm	Recall		
	600 APK	7000 APK	14170 APK
K-Nearest Neighbour (KNN)	88%	85.2%	88%
Decision Tree (DT)	100%	88.8%	91.2%
Support Vector Machine (SVM)	97%	89.4	90.6%

There is a decrease in performance for the model generated from the K-Nearest Neighbor algorithm, Support Vector Machine and Decision Tree. Table IV, Accuracy of KNN, Support Vector and Decision Tree classifier decreased when using a larger dataset. This is because the three algorithms are suitable for use if the dataset is small. The larger the size of the training dataset, the lower the accuracy. Table V Precision decreased if the classifier was

carried out with the three non-neural network algorithms. Table VI F1-Score experienced a decrease in the classification of large datasets and Table VII Recall decreases if the dataset is large. In this study, the use of a large dataset is not suitable when using a large dataset, and it is tried to train the dataset using a Neural Network.

C. RQ 3, What is the percent increase in accuracy with the implementation of the Multi-Layer Perceptron algorithm?

TABLE VII. PERFORMANCE FOR ALGORITHM DEEP LEARNING

Performance	Dataset		
	600 APK	7000 APK	14170 APK
Accuracy	99%	100%	100%
Precision	99%	100%	100%
Recall	99%	100%	100%
F1-Score	99%	100%	100%

The results of the Multi-Layer Perceptron classification experiment show that performance increases with increasing datasets. The more the number of datasets, the better for performance. Experiment from dataset 600 APK = 99%, dataset 7000 APK = 100% and dataset 14170 APK = 100%.

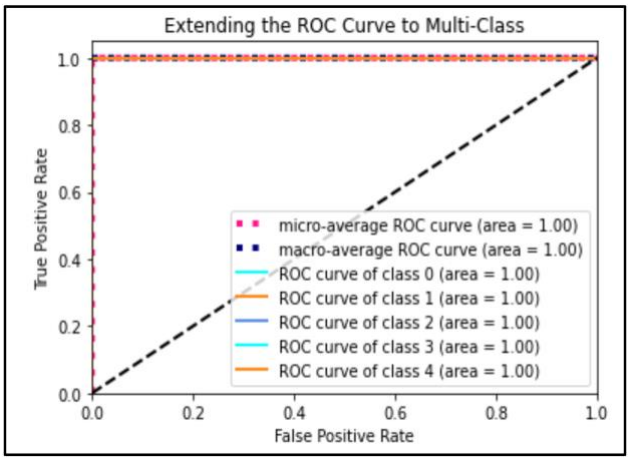

Fig. 4. ROC Multi-Class Multi-Layer Perceptron Algorithm

Receiver Operating Characteristic (ROC) is a plot of True Positive Rate (TPR) on the y-axis and False Positive Rate (FPR) on the x-axis. Where, True Positive Rate = True Positives / (True Positives + False Negatives) and False Positive Rate = False Positives / (False Positives + True Negatives). It can be seen that the ROC and Area Under Curve scores show significant values.

In Figure 4, the ROC of the model results from the Artificial Neural Network Classifier from the malware dataset. ROC (Receiver Operating Characteristics) is a performance measurement tool for classification problems in determining the threshold of the model. Malware Banking APK APK file label 0, symbolized in light blue. APK file Malware Ransomware APK label 1, symbolized in orange. APK file Malware Riskware APK label 2, symbolized in blue. Malware SMS APK file APK label 3, symbolized in light blue. APK file Malware Benign APK label 4, symbolized in orange. The y-axis represents the True Positive Rate (sensitivity), the x-axis represents the False Positive Rate (Specificity). Figure 4 shows the higher the True Positive Rate (sensitivity) and the smaller the False Positive Rate, the better the threshold. The optimistic Area Under Curve (AUC) value from the Artificial Neural Network validation results shows a value of = 1. This shows that the accuracy results obtained are in the very good category.

D. RQ 4, Is it effective to perform malware analysis using static methods?

Using this static method does not require running the malware into an isolated or controlled environment. The malware APK file is only extracted, then stored into the malware dataset. The dataset is classified using the classification method and then the model is tested with the extracted malware dataset. The results are effective for detecting the Android APK file is infected with malware or normal. The static method is actually simple and works effectively in malware detection.

VI. CONCLUTION

Based on the results of experiments conducted in this study, it can be concluded that classification using machine learning produces good accuracy on the K-Nearest Neighbor algorithm, Support Vector Machine and Decision Tree. However, the use of larger datasets causes a decrease in accuracy. This factor causes the use of deep learning in training datasets in order to produce high accuracy on large datasets. The accuracy of the K-Nearest Neighbor algorithm on average = 88%, if using the 14170 APK dataset. Average Support Vector Machine accuracy = 90.5%, when using the 14170 APK dataset. Average Decision Tree accuracy = 90.8%, when using the 14170 APK dataset. Accuracy using deep learning with Multi-Layer Perceptron results in 100% accuracy, using the 14170 APK dataset.

REFERENCES

[1] A. Ghasempour, N. Fazlida, M. Sani, and O. J. Abari, "Permission Extraction Framework for Android Malware Detection," vol. 11, no. 11, pp. 463–475, 2020.

[2] O. N. Elayan and A. M. Mustafa, "Android malware detection using deep learning," *Procedia Comput. Sci.*, vol. 184, no. 2019, pp. 847–852, 2021, doi: 10.1016/j.procs.2021.03.106.

[3] S. R. T. Mat, M. F. A. Razak, M. N. M. Kahar, J. M. Arif, and A. Firdaus, "A Bayesian probability model for Android malware detection," *ICT Express*, no. xxxx, 2021, doi: 10.1016/j.icte.2021.09.003.

[4] M. Qiao, A. H. Sung, and Q. Liu, "Merging permission and api features for android malware detection," *Proc. - 2016 5th IIAI Int. Congr. Adv. Appl. Informatics, IIAI-AAI 2016*, pp. 566–571, 2016, doi: 10.1109/IIAI-AAI.2016.237.

[5] A. K. Singh, C. D. Jaidhar, and M. A. A. Kumara, "Experimental analysis of Android malware detection based on combinations of permissions and API-calls,"

J. Comput. Virol. Hacking Tech., vol. 15, no. 3, pp. 209–218, 2019, doi: 10.1007/s11416-019-00332-z.

[6] M. Shohel Rana, C. Gudla, and A. H. Sung, "Evaluating machine learning models for android malware detection - A comparison study," *ACM Int. Conf. Proceeding Ser.*, pp. 17–21, 2018, doi: 10.1145/3301326.3301390.

[7] X. Wang, D. Zhang, X. Su, and W. Li, "Mlifdect: Android malware detection based on parallel machine learning and information fusion," *Secur. Commun. Networks*, vol. 2017, 2017, doi: 10.1155/2017/6451260.

[8] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "ANASTASIA: ANdroid mAlware detection using STatic analySIs of applications," *2016 8th IFIP Int. Conf. New Technol. Mobil. Secur. NTMS 2016*, 2016, doi: 10.1109/NTMS.2016.7792435.

[9] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection," *Comput. Secur.*, vol. 65, no. March, pp. 121–134, 2017, doi: 10.1016/j.cose.2016.11.007.

[10] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, and Y. Rahulamathavan, "PIndroid: A novel Android malware detection system using ensemble learning methods," *Comput. Secur.*, vol. 68, pp. 36–46, 2017, doi: 10.1016/j.cose.2017.03.011.

[11] A. Abusitta, M. Q. Li, and B. C. M. Fung, "Malware classification and composition analysis: A survey of recent developments," *J. Inf. Secur. Appl.*, vol. 59, no. April, p. 102828, 2021, doi: 10.1016/j.jisa.2021.102828.

[12] V. Syrris and D. Geneiatakis, "On machine learning effectiveness for malware detection in Android OS using static analysis data," *J. Inf. Secur. Appl.*, vol. 59, no. May, p. 102794, 2021, doi: 10.1016/j.jisa.2021.102794.

[13] P. Opěla, I. Schindler, P. Kawulok, R. Kawulok, S. Rusz, and H. Navrátil, "On various multi-layer perceptron and radial basis function based artificial neural networks in the process of a hot flow curve description," *J. Mater. Res. Technol.*, vol. 14, pp. 1837–1847, 2021, doi: 10.1016/j.jmrt.2021.07.100.

[14] M. L. Baptista, E. M. Elsa, and K. Goebel, "A self-organizing map and a normalizing multi-layer perceptron approach to baselining in prognostics under dynamic regimes," *Neurocomputing*, vol. 456, pp. 268–287, 2021, doi: 10.1016/j.neucom.2021.05.031.

[15] M. Amin, T. A. Tanveer, M. Tehseen, M. . Khan, F. A. Khan, and S. Anwar, "Static malware detection and attribution in android byte-code through an end-to-end deep system," *Futur. Gener. Comput. Syst.*, vol. 102, pp. 112–126, 2020, doi: 10.1016/j.future.2019.07.070.

[16] L. Xiong and Y. Yao, "Study on an adaptive thermal comfort model with K-nearest-neighbors (KNN) algorithm," *Build. Environ.*, vol. 202, no. May, p. 108026, 2021, doi: 10.1016/j.buildenv.2021.108026.

[17] D. Sisodia and D. S. Sisodia, "Quad division prototype selection-based k-nearest neighbor classifier for click fraud detection from highly skewed user click dataset," *Eng. Sci. Technol. an Int. J.*, no. xxxx, 2021, doi: 10.1016/j.jestch.2021.05.015.

[18] A. Shokrzade, M. Ramezani, F. Akhlaghian Tab, and M. Abdulla Mohammad, "A novel extreme learning machine based kNN classification method for dealing with big data," *Expert Syst. Appl.*, vol. 183, no. May, p. 115293, 2021, doi: 10.1016/j.eswa.2021.115293.

[19] X. Zhu, C. Ying, J. Wang, J. Li, X. Lai, and G. Wang, "Ensemble of ML-KNN for classification algorithm recommendation," *Knowledge-Based Syst.*, vol. 221, p. 106933, 2021, doi: 10.1016/j.knosys.2021.106933.

[20] C. E. A. Bundak, M. A. Abd Rahman, M. K. Abdul Karim, and N. H. Osman, "Fuzzy rank cluster top k Euclidean distance and triangle based algorithm for magnetic field indoor positioning system," *Alexandria Eng. J.*, 2021, doi: 10.1016/j.aej.2021.08.073.

[21] R. Taheri, M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti, "Similarity-based Android malware detection using Hamming distance of static binary features," *Futur. Gener. Comput. Syst.*, vol. 105, pp. 230–247, 2020, doi: 10.1016/j.future.2019.11.034.

[22] R. Suwanda, Z. Syahputra, and E. M. Zamzami, "Analysis of Euclidean Distance and Manhattan Distance in the K-Means Algorithm for Variations Number of Centroid K," *J. Phys. Conf. Ser.*, vol. 1566, no. 1, 2020, doi: 10.1088/1742-6596/1566/1/012058.

[23] S. Zhang, "Cost-sensitive KNN classification," *Neurocomputing*, vol. 391, no. xxxx, pp. 234–242, 2020, doi: 10.1016/j.neucom.2018.11.101.

[24] T. Du, Z. Zhao, Q. Zhu, and L. Tian, "Locating a γ-ray source using cuboid scintillators and the KNN algorithm," *Nucl. Instruments Methods Phys. Res. Sect. A Accel. Spectrometers, Detect. Assoc. Equip.*, vol. 993, no. January, 2021, doi: 10.1016/j.nima.2021.165069.

[25] R. M. Arias Velásquez, "Support vector machine and tree models for oil and Kraft degradation in power transformers," *Eng. Fail. Anal.*, vol. 127, no. May, 2021, doi: 10.1016/j.engfailanal.2021.105488.

[26] Y. Arbabi Yazdi, H. Toossian Shandiz, and H. Gholizadeh Narm, "Stiction detection in control valves using a support vector machine with a generalized statistical variable," *ISA Trans.*, no. xxxx, 2021, doi: 10.1016/j.isatra.2021.07.020.

[27] J. Lesouple, C. Baudoin, M. Spigai, and J. Y. Tourneret, "How to introduce expert feedback in one-class support vector machines for anomaly detection?," *Signal Processing*, vol. 188, p. 108197, 2021, doi: 10.1016/j.sigpro.2021.108197.

[28] X. Ju, Y. Tian, D. Liu, and Z. Qi, "Nonparallel hyperplanes support vector machine for multi-class classification," *Procedia Comput. Sci.*, vol. 51, no. 1, pp. 1574–1582, 2015, doi: 10.1016/j.procs.2015.05.287.

[29] Q. Zhang, H. Wang, and S. W. Yoon, "A 1-norm regularized linear programming nonparallel hyperplane support vector machine for binary classification problems," *Neurocomputing*, vol. 376, no. xxxx, pp. 141–152, 2020, doi: 10.1016/j.neucom.2019.09.068.

[30] Z. Zhao, P. Zhong, and Y. Zhao, "Learning SVM with weighted maximum margin criterion for classification of imbalanced data," *Math. Comput. Model.*, vol. 54, no. 3–4, pp. 1093–1099, 2011, doi: 10.1016/j.mcm.2010.11.040.

[31] J. Mariéthoz and S. Bengio, "A kernel trick for sequences applied to text-independent speaker verification systems," *Pattern Recognit.*, vol. 40, no. 8, pp. 2315–2324, 2007, doi: 10.1016/j.patcog.2007.01.011.

[32] S. F. Hussain, "A novel robust kernel for classifying high-dimensional data using Support Vector Machines," *Expert Syst. Appl.*, vol. 131, pp. 116–131, 2019, doi:

10.1016/j.eswa.2019.04.037.

[33] X. Huang, A. Maier, J. Hornegger, and J. A. K. Suykens, "Indefinite kernels in least squares support vector machines and principal component analysis," *Appl. Comput. Harmon. Anal.*, vol. 43, no. 1, pp. 162–172, 2017, doi: 10.1016/j.acha.2016.09.001.

[34] M. Moshkov, "Decision trees based on 1-consequences," *Discret. Appl. Math.*, vol. 302, pp. 208–214, 2021, doi: 10.1016/j.dam.2021.07.017.

[35] V. Gumuskaya, W. van Jaarsveld, R. Dijkman, P. Grefen, and A. Veenstra, "Integrating stochastic programs and decision trees in capacitated barge planning with uncertain container arrivals," *Transp. Res. Part C Emerg. Technol.*, vol. 132, no. December 2020, p. 103383, 2021, doi: 10.1016/j.trc.2021.103383.

[36] A. Strzelecka and D. Zawadzka, "Application of classification and regression tree (CRT) analysis to identify the agricultural households at risk of financial exclusion," *Procedia Comput. Sci.*, vol. 192, pp. 4532–4541, 2021, doi: 10.1016/j.procs.2021.09.231.

[37] D. H. Lee, S. H. Kim, and K. J. Kim, "Multistage MR-CART: Multiresponse optimization in a multistage process using a classification and regression tree method," *Comput. Ind. Eng.*, vol. 159, no. May, p. 107513, 2021, doi: 10.1016/j.cie.2021.107513.

[38] M. Li, P. Vanberkel, and X. Zhong, "Predicting ambulance offload delay using a hybrid decision tree model," *Socioecon. Plann. Sci.*, no. July, p. 101146, 2021, doi: 10.1016/j.seps.2021.101146.

[39] W. Gao, Z. Bai, F. Zhu, C. C. Chou, and B. Jiang, "A study on the cyclist head kinematic responses in electric-bicycle-to-car accidents using decision-tree model," *Accid. Anal. Prev.*, vol. 160, no. May 2020, p. 106305, 2021, doi: 10.1016/j.aap.2021.106305.