ISSN 2409-2665
Journal of Logistics, Informatics and Service Science
Vol. 11 (2024) No. 10, pp. 1-19
DOI:10.33168/JLISS.2024.1001

Android Malware Multiclass Classification using Machine Learning: Evaluating the Performance of Random Forest, Artificial Neural Network, and Convolutional Neural Network

Yoon-Teck Bau[1], Yan-Hui Choo[*] and Chien-Le Goh[1]

[1] Multimedia University, Persiaran Multimedia, 63100 Cyberjaya, Malaysia

chooyanhui1717@gmail.com/1201303209@student.mmu.edu.my,

ytbau@mmu.edu.my (Corresponding author), clgoh@mmu.edu.my

Abstract. This research investigates the effectiveness of machine learning techniques, namely random forest, artificial neural network, and convolutional neural network, in detecting and classifying Android malware using both static and dynamic analysis methods. Leveraging the CICInvesAndMal2019 dataset, multiclass classification by categorizing malware into adware, ransomware, scareware, and SMS malware are performed. The static analysis examines permissions and intents, while the dynamic analysis focuses on API calls and network flows. The performance of the models is evaluated using accuracy, recall, precision, F1 score, training time, and testing time. The results demonstrate the superiority of random forest over deep learning models in both static and dynamic analysis, with static analysis yielding better performance than dynamic analysis. This study contributes to the field of Android malware detection by providing insights into the effectiveness of different machine learning algorithms and analysis methods, highlighting the potential of random forest for efficient and accurate malware multiclass classification.

Keywords: Mobile Phone OS, Malware Analysis, Machine Learning, Android OS, Multiclass Classification, Artificial Neural Network, Convolutional Neural Network, Random Forest, Static Analysis, Dynamic Analysis

1. Introduction

In recent years, the rapid growth of mobile devices, particularly those running on the Android operating system has been into a revolutionary surge. With this widespread adoption, the mobile ecosystem has become a part of our daily lives, facilitating communication, productivity, and entertainment. However, the trend of mobile applications and the high number of Android customers has also attracted the attention of cybercriminals, leading to an escalation in the sophistication and prevalence of Android malware (Selvaganapathy et al., 2021).

The Android platform encountered over 4.18 million malicious applications in 2019 alone, with a daily emergence of approximately 11,500 new instances of Android malware. The trend is also expected to increase in the future (Haidros Rahima Manzil & Manohar Naik, 2024). Malicious actors develop advanced strategies to exploit vulnerabilities regularly, posing major dangers to user privacy and security. The impact of Android malware on user privacy and security is profound, manifesting in various forms such as data theft, financial fraud, and device compromise. For instance, malware can steal sensitive information like login credentials and personal data, leading to identity theft and unauthorized access to accounts. Financial fraud is another critical issue, with malware often used to conduct unauthorized transactions or steal payment information. Additionally, device compromise can occur, allowing attackers to control and manipulate infected devices, leading to significant disruptions and potential further exploitation.

The growing volume and diversity of malware pose significant challenges in effective detection (Selvaganapathy et al., 2021). Addressing this issue has led to the exploration of artificial intelligence, with a specific focus on machine learning to enhance the effectiveness of malware detection and analysis. The rise of malware applications has made it impractical to manually create detection rules in today's cybersecurity landscape. Recognizing this challenge, researchers and commercial anti-malware products have increasingly turned to the application of machine learning to fortify their defense mechanisms. Malware analysis using machine learning can learn complex patterns of malware, enables machine learning based systems to identify differences in existing malware and identify new and unseen malwares (Muzaffar et al., 2022). As a subset of machine learning, deep learning is also widely used.

Other than signature-based approach, the behavior-based approaches encompass static analysis, dynamic analysis, and hybrid analysis. Static analysis is the process of collecting static features without launching applications, utilizing tools to capture information from the Android package files like AndroidManifest.xml, metadata, and classes. This method is computationally efficient and preferable due to its simplicity. On the other hand, dynamic analysis executes the applications on a device or emulator to capture features like application programming interface logs during runtime. Furthermore, researchers often opt for the hybrid analysis framework, combining the static methods and dynamic methods. Hybrid analysis leverage the strengths of both approaches to enhance detection accuracy, demonstrating the evolving landscape of Android malware research (Haidros Rahima Manzil & Manohar Naik, 2024).

In the current landscape of Android malware detection, machine learning based Android malware detection has been primarily focusing on binary classification, distinguishing between malicious and benign Android applications, such as HR (2019), Hadiprakoso et al. (2020), and Anuar et al. (2020). However, this concentration has inadvertently led to a lack of research on multiclass Android malware classification, particularly concerning deep learning methodologies, as noted by Wang et al. (2020). The absence of exploration in multiclass classification restricts our understanding of the diverse spectrum of Android malware and limits the development of effective mitigation strategies. Multiclass classification categorizes malware into distinct classes, offers a way to gain deeper insights into the specific behaviors, characteristics, and potential risks associated with each category, thereby enabling more targeted and robust security measures.

The present research landscape also directs its attention towards static analysis for Android malware

classification, such as Almahmoud et al. (2021), Shatnawi et al. (2022), and Kabakus (2022). Only a limited amount of research conducts dynamic or hybrid analysis for malware detection in Android system. While static analysis is efficient in detecting malware, its limitations become evident when confronted with applications employing methods to obscure and protect their code like encryption. In contrast, dynamic analysis offers a feasible way to thoroughly examine malware (Wang et al., 2020). By focusing on dynamic features like runtime interactions and system calls, it enhances the ability to adapt to emerging malware threats and contributes to a more robust and proactive defense strategy.

This research aims to explore the effectiveness of Android malware multiclass classification using machine learning. Specifically, to investigate the performance of random forest (RF), artificial neural network (ANN), and convolutional neural network (CNN) models for Android malware multiclass classification using static and dynamic analysis. RF is known for its robustness and interpretability in handling structured data, ANN provides flexibility and adaptability in learning complex patterns, and CNN excels in feature extraction and hierarchical learning, particularly in image and sequence data, which can be useful to extracting features from Android malware analysis.

2. Literature Review

2.1. Datasets for Android Malware Analysis

Datasets with large malware samples are a crucial part in the realm of malware analysis, particularly when leveraging machine learning techniques. These datasets consist of large amounts of adware, ransomware, benign, and malware samples. These datasets serve as valuable resources for researchers, providing labelled data for supervised learning and facilitating the accessibility of malware samples for analysis. The following section will delve into the characteristics of three reviewed datasets.

The first dataset worth mentioning is the Drebin dataset. According to Mbunge et al. (2023), the Drebin dataset is widely utilized in the analysis of Android malware. It comprises 5560 samples belonging to 179 malware families, collected during the period from 2010 to 2012. Despite its age, Drebin dataset is still used in recent research. For example, Fiky et al. (2021) combined Drebin and Malgenome datasets and obtained a dataset with 6,820 malware samples and 12,015 benign samples. This dataset was characterized by a rich feature set, encompassing 215 features for a comprehensive analysis. Raghuvanshi & Singh (2023) and Sarah et al. (2021) also extracted 215 features from Drebin dataset. However, it is also highlighted that the dynamic nature of malware characteristics over time posed a challenge to deep learning models when utilizing Drebin (Mbunge et al., 2023).

The Android malware datasets accessible from Canadian Institute for Cybersecurity (CIC) are also widely used. The CIC-AndMal2017 dataset consists of 426 malware samples obtained from Virustotal service, Contagio security blog and 5,065 benign samples obtained from Google Play. The CIC-AndMal2017 dataset incorporates features like memory dump, network traffic, logs, application programming interface (API) calls, permissions, and phone statistics data, obtained through the installation of malware on real devices. This approach directly addresses a limitation observed in other datasets, such as Drebin, where data is not obtained from real devices (Lashkari et al., 2018).

Subsequently, the CIC-InvesAndMal2019 published in 2019 enriches the CIC-AndMal2017 dataset by adding new features, namely permissions & intents and API calls. CIC-InvesAndMal2019 is the second part of CIC-AndMal2017 dataset, same as CIC-AndMal2017 dataset, it consists of 426 malware samples. The malware samples of both datasets are also categorized into adware, ransomware, scareware, and SMS malware (Taheri et al., 2019).

2.2. Android Malware Analysis using Machine Learning

Research from Raghuvanshi & Singh (2023), proposed a method for identifying malicious and benign applications on Android devices. The paper employs a static analysis approach based on four types of features extracted from Android Application Package (APK) files: permissions, API calls, intents, and command signatures. Two malware datasets, Drebin and Malgenome, were used in the training and

testing process of three machine learning classifiers, which are support vector machine (SVM), maximum voting classifier (MVC), and random forest (RF). Principal component analysis (PCA) was also applied to optimize the features and reduce the dimensionality. For feature extraction, the paper uses AndroGuard, a tool for decompiling APK files, to obtain the manifest.xml and class.dex files, and extracted 215 features from these files, including 113 permissions and 102 features of API calls, intents, and commands. To facilitate experimentation, two distinct feature sets are generated. The first feature set referred to as the “combined feature set”, comprises all 215 features extracted from the manifest and class files. Meanwhile, the second feature set, referred to as the "permission set", exclusively includes the binary combinations of the 113 permissions. PCA was used separately for the combined feature set. The permission feature set was to reduce the number of features. The results using the combined features showed that the SVM classifier with PCA achieves the highest accuracy of 98.19% on the Drebin dataset and 98.84% for the Malgenome dataset, while SVM classifier without PCA achieves 98.03% and 98.70% on the Drebin and Malgenome datasets, respectively. However, the results using the permission features showed that the RF classifier achieves the highest accuracy of 96.27% on the Drebin dataset while SVM classifier with PCA achieves the highest 97.63% on the Malgenome dataset. Both machine learning trained solely on permission features exhibit lower performance compared to those utilizing the comprehensive set of combined features. This discrepancy suggests that permissions alone may not be sufficient for effectively classifying malware and benign software.

Sarah et al. (2021) presented an approach for binary classification by classifying malware and benign classes using ensemble machine learning algorithms and feature selection methods. Drebin datasets were used and the performance of eight machine learning algorithms, including four traditional ones which are logistic regression, support vector machine, gaussian naive bayes, and decision tree and four ensemble ones namely, random forest, gradient boosting decision tree, light GBM, and XGBoost were compared. recursive feature elimination (RFE) and recursive feature elimination with cross-validation (RFECV) were applied to select the optimal subset of features for each algorithm. Their research uses two approaches for model testing, namely train-test split and cross validation. The results show that ensemble machine learning algorithms achieve higher accuracy than traditional ones for predicting Android malware. Among the ensemble methods, lightGBM has the best performance, with an accuracy of 99.50% using cross-validation and 99.30% using train-test split. This is because lightGBM splits the tree leaf-wise, which can reduce overfitting and improve efficiency. Their research also used RFE to select different numbers of features, ranging from 15 to 100 to compare the accuracy of the models. LightGBM and random forest have the highest accuracy with 100 and 55 features, respectively. The authors also use RFECV to automatically select the optimal number of features for each algorithm and find that lightGBM and decision tree have the best performance with 100 and 55 features, respectively. Their research mentioned that for higher accuracy, lightGBM is a stronger choice, excelling in complex datasets. However, if efficiency and fewer features are prioritized, random forest consistently delivers competitive accuracy with better performance.

Bayazit et al. (2022) implemented recurrent neural network (RNN), gated recurrent unit (GRU), long short-term memory (LSTM), and bidirectional long short-term memory (Bi-LSTM) to detect malware in Android system using static analysis. In their research, the CICInvesAndMal2019 dataset was employed to detect Android malware. The study categorized two distinct categories by labelling the malware in the dataset, namely ransomware, adware, scareware, and SMS malware as 1, while benign samples are labelled as 0. During the classification of deep learning, the vanishing gradient problem poses a limitation for RNN, hindering their ability to effectively capture long-term dependencies in data. In response to this limitation, the LSTM model was employed in their research. LSTM, categorized as a type of RNN, incorporate a “memory cell” capable of retaining information for extended durations, thereby mitigating the challenges associated with the vanishing gradient problem. However, LSTMs exhibit a unidirectional data processing approach, implying an inability to capture patterns from future steps in the sequence, resulting in underestimation. To address this constraint, a

transition was made to the BiLSTM model. BiLSTM process data in both forward and backward directions, enabling the extraction of patterns from both preceding and subsequent data points. The four deep learning models achieved accuracies of 98.02%, 98.10%, 98.75%, and 98.85% for RNN, GRU, LSTM, and BiLSTM, respectively. It is also mentioned that their future work will focus on reducing the train time for the deep learning models.

In the research from Imtiaz et al. (2021), they introduced a deep learning model named deep Android malware detection (DeepAMD) that uses artificial neural network (ANN) for Android malware classification. To evaluate the effectiveness of their proposed model, the researchers compared four traditional machine learning algorithms, namely multilayer perceptron (MLP), sequential minimal optimization (SMO), naïve Bayes (NB), and decision tree (DT/J48). Similar to the approach taken by Bayazit et al. (2022), they utilized the CICInvesAndMal2019 dataset in their research. This dataset includes labeled samples with information about malware categories and families. In their investigation, the researchers initially performed binary classification on the static layer of the dataset, focusing on static features of the applications. Upon classifying samples as malware, they subsequently delved into further categorization, distinguishing between four malware categories: adware, ransomware, SMS malware, and scareware. Additionally, they extended their analysis to classify malware samples into a total of 39 distinct families, encompassing variations within each of the four categories. In binary classification, their model achieved an accuracy of 93.40%. The static layer has an accuracy of 92.50% in malware category classification. However, the dynamic layer presented a lower accuracy of 80.30%, underscoring the distinctions in performance between static and dynamic analysis. During family classification, their proposed model achieved an overall accuracy of 90.00% on static layer. However, their model only achieved 55.70% on malware family classification using dynamic layer. This result is worse than the NB machine learning algorithm, which recorded an accuracy of 59.00% in the same context. This discrepancy emphasizes that traditional machine learning approaches may outperform deep learning models in efficiency or accuracy.

The study by Shyong et al. (2020), introduces an approach that combines static analysis and dynamic analysis for malware detection and family classification. The researchers implemented their methodology within the Android application analysis system. During static analysis, the permissions list of the application is extracted to facilitate binary malware classification. This process serves to discern whether the application exhibits characteristics indicative of malware or benign intent. If the application is detected as malware during static analysis, their system progresses to dynamic analysis. At this stage, various dynamic features are extracted from the network packets associated with the malicious application. These features include DNS packet features, HTTP packet features, TCP packet features, and relevant information derived from the network packets file. The dynamic analysis utilizes the extracted network-related features to classify the detected malware into specific families, with a total of 10 malware families and a benign family. Their study utilizes random forest (RF) machine learning algorithm for the classification tasks in both static and dynamic analysis. The performance of five machine learning algorithms during static analysis are evaluated and RF achieved the highest accuracy among them. The accuracy of random forest using different feature sets were explored in this study. Feature set comprising of 226 features obtained from the system permissions achieved an accuracy of 98.86%, while feature set with 2582 features comprising of system permissions and custom permissions achieved a lower accuracy of 98.72%. Furthermore, their study identified the optimal parameter for the random forest algorithm, which is the most effective number of trees is either 300 or 400. The random forest algorithm in their paper also achieved 95.60% for dynamic analysis which includes TCP packet, DNS packet, HTTP packet, and other contents as the features.

2.3. Research Gaps

While numerous studies have addressed the binary classification problem of distinguishing between malicious and benign applications using machine learning techniques, there exists a research gap in the

exploration of multiclass malware classification scenarios. A considerable proportion of existing literature only focuses on binary classification, where the primary goal is to identify whether an Android package file is malicious or benign. This trend is evident in several studies, such as HR (2019), Hadiprakoso et al. (2020), and Anuar et al. (2020), where researchers have primarily concentrated on the binary classification of Android applications. For instance, in a recent study by Bayazit et al., (2022), the researchers investigated malware detection on Android applications using the CICInvesAndMal2019 dataset, which classified malware samples into four categories, namely adware, ransomware, scareware, and SMS malware. Despite the availability of labelled malware samples spanning different classes, the authors opted for a binary classification approach, merging the various malware categories into a single "malware" class. Their primary objective was to discern between the aggregated malware class and benign Android applications using deep learning models. While such an approach can be used for identifying the presence or absence of malware, it lacks the ability required for an in-depth analysis of the distinct threats posed by different malware categories. This research emphasizes the need for multiclass approaches capable of capturing the distinctions among various malware categories.

Despite the growing of machine learning models in classifying Android applications, a research gap exists in addressing the temporal dynamics of malware evolution. Dynamic analysis involves observing an application's behavior during execution, allowing for a real-time understanding of its actions and interactions with the devices and the networks. Many existing studies focus on training models on static typed dataset, prioritizing attributes extracted from the code of Android applications without sufficiently considering the dynamic aspects of malware behaviors. Static features often rely on characteristics such as permissions, API calls, and manifest information during the model training phase. Research solely using static analysis often overlooking the insights that dynamic analysis can offer. For instance, the work by Subash A et al. (2023) explored Android malware detection exclusively through static analysis, leveraging features derived from the manifest file and application permissions. Similarly, Almahmoud et al. (2021) extracted four static features from the manifest file, namely permissions, API calls, system events and permission rate to discern between benign and malicious applications, ignoring the insights in dynamic behaviors. Additionally, in the research from Shatnawi et al. (2022), they obtained API calls and permission features from the CICInvesAndMal2019 dataset to detect malware in Android using machine learning. These examples highlight the prevailing inclination towards static analysis in the current landscape of Android malware detection research, further emphasizing the need to incorporate dynamic analysis methodologies. Table 1 shows the other reviewed papers with their analysis type and classification method.

Table 1: Analysis type and classification method of other papers

Reference	Analysis Type	Classification
Amenova et al., 2022	Static	Binary
Raghuvanshi & Singh, 2023	Static	Binary
Elayan & Mustafa, 2021	Static	Binary
Sarah et al., 2021	Static	Binary
Eltaher et al., 2023	Dynamic	Binary
Yang et al., 2022	Static	Multiclass
Kabakus, 2022	Static	Binary
Mohamad Arif et al., 2021	Static	Binary
Haidros Rahima Manzil & Manohar Naik, 2022	Dynamic	Binary
Ibrahim et al., 2022	Static	Multiclass

3. Research Methods

3.1. Dataset

In this research, the Canadian Institute for Cybersecurity Investigation of the Android Malware 2019

(CICInvesAndMal2019) datasets were used. These datasets are distinguished by their separation of static and dynamic features, which enables a two-pronged analysis approach. The malware samples were collected from diverse sources including VirusTotal, Contagio security blog, and contributions from previous researchers in the field. In Figure 1, the static dataset includes 1594 samples (1187 benign and 407 malware), with 8115 columns. Four columns which are "family", "category", "MD5", and "binary type" serve as identifiers while the remaining 8111 columns are attributes extracted from application permissions and intents. Static dataset consists of five classes, namely benign, adware, ransomware, SMS malware, and scareware.

Figure 2 is the dynamic dataset, which encompasses 327 samples and consists of 920 columns. Within the dataset, 918 columns are attributes derived from API calls and network flows while the remaining two columns which are "ID" and "category" serve as the identifiers. Dynamic dataset has four classes by categorizing malware into adware, ransomware, SMS malware, and scareware.

While this dataset offers a comprehensive collection of Android malware samples and their respective features, using a single dataset may not fully capture the diversity and variability of malware found in the real world. Consequently, the performance metrics obtained in this study might be specific to this dataset and may not generalize well to other datasets with different distributions, feature sets, or malware types.

	<actionandro...	<actionandro...	<actionandro...	<actionandro...	<actionandro...	...	com.flyersof...	<family>	<category>	<MD5>	Binary_Type
0	0	0	0	0	0	...	0	dowgin	Adware	1c4e357a8ec...	Malware
1	0	0	0	0	0	...	0	dowgin	Adware	1d15765ffee...	Malware
2	0	0	0	0	0	...	0	dowgin	Adware	31c657bf77e...	Malware
3	0	0	0	0	0	...	0	dowgin	Adware	37b993b5f59...	Malware
4	0	0	0	0	0	...	0	dowgin	Adware	3e30f2644a2...	Malware
...	...	...	...	...	...	...	...	...	...	...	...
1589	0	0	0	0	0	...	0	plankton	SMS	0e3294e4ff1...	Malware
1590	0	0	0	0	0	...	0	smssniffer	SMS	af1863dd2ba...	Malware
1591	0	0	0	0	0	...	0	smssniffer	SMS	b1ae0d9a279...	Malware
1592	0	0	0	0	0	...	0	smssniffer	SMS	c9e3af6a442...	Malware
1593	0	0	0	0	0	...	0	smssniffer	SMS	e9068f11699...	Malware

[1594 rows x 8115 columns]

Fig. 1: Static Dataset

	id	2Grammed_APICalls_#0	2Grammed_APICalls_#1	2Grammed_APICalls_#2	2Grammed_APICalls_#3	...	Idle_Mean	Idle_Std	Idle_Max	Idle_Min	<Category>
0	1	0	42	0	0	...	170240589	19488345	186397186	155416258	ransomware
1	2	2	0	0	0	...	166045107	8283170	173387076	159763065	ransomware
2	3	0	40	0	0	...	246965228	22019770	263938727	230197339	ransomware
3	4	0	43	0	0	...	218034819	31463640	242915705	194688432	ransomware
4	5	2	43	0	0	...	200495551	20186565	218041946	185179596	ransomware
...	...	...	...	...	...	...	...	...	...	...	...
322	112	0	2	0	0	...	41903090	9253128	49854310	34664042	smsMalware
323	113	0	0	0	0	...	30594702	6345981	35835129	25532067	smsMalware
324	114	0	0	0	0	...	34409655	2404418	36331095	32462336	smsMalware
325	115	0	0	0	0	...	37266333	3018065	39871270	34911092	smsMalware
326	116	0	0	0	0	...	223758200	22179803	241166832	206910986	smsMalware

327 rows × 920 columns

Fig. 2: Dynamic Dataset

3.2. Data Analysis

In the static dataset, instances are categorized into five classes as shown in Figure 3. There are a total of 1187 benign samples. 407 malware samples are distributed into 111 instances of scareware, 108 instances of SMS malware, 95 instances of adware, and 93 instances of ransomware.

In the dynamic dataset, instances comprise only malware samples as illustrated in Figure 4. The dataset contains 98 instances of scareware, 86 instances of SMS malware, 75 instances of adware, and 68 instances of ransomware. Unlike the static dataset, the absence of benign samples in the dynamic dataset highlights its focus on malicious behavior.

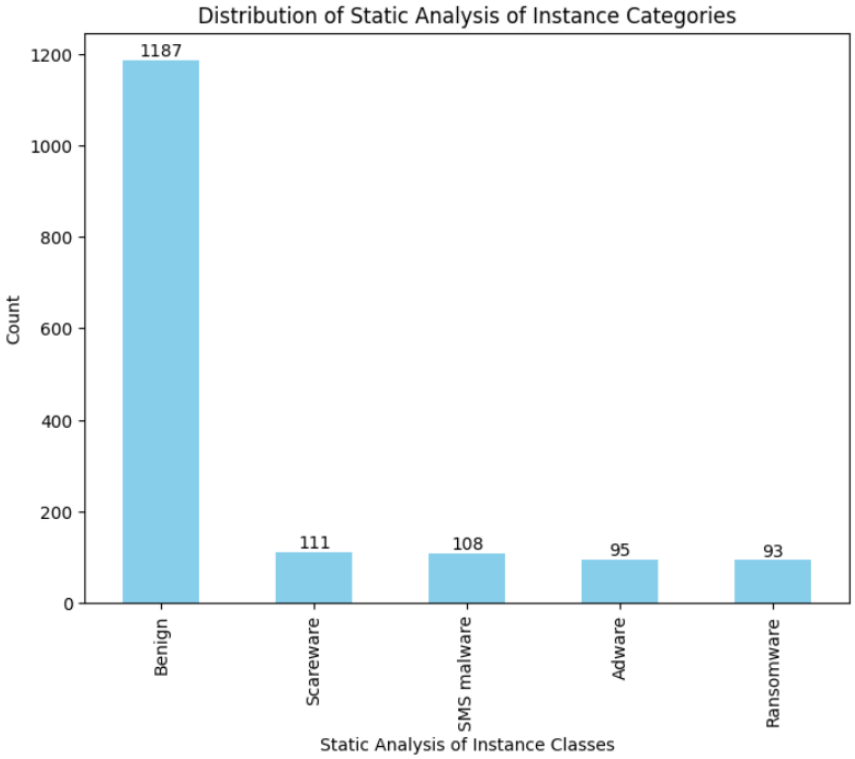

Fig. 3: Distribution of Static Analysis of Instance Categories

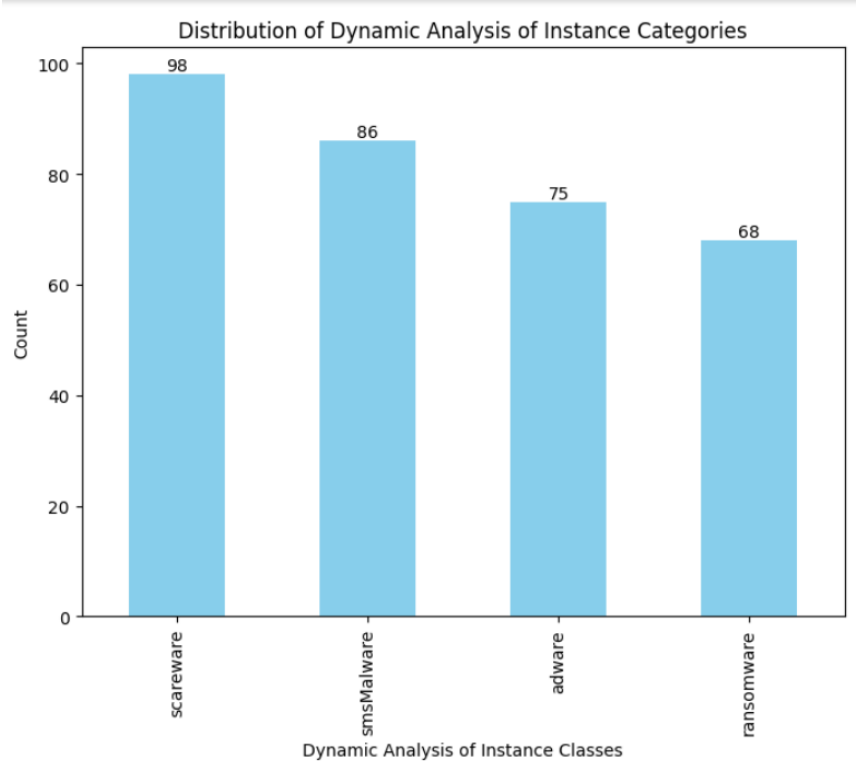

Fig. 4: Distribution of Dynamic Analysis of Instance Categories

Within the static dataset, 3996 columns pertain to intents, while the remaining 4115 columns correspond to permissions. These features are obtained from the Android package file (APK). Intents facilitate communication between different parts of an app or even between apps themselves. Permissions, on the other hand, represent access rights that apps request to perform specific actions on

the device or user data. Both intents and permissions columns function as a single feature, holding a value of either 1 or 0. A value of 1 signifies that the app declares a specific intent action like launching a camera, or requests a particular permission like accessing the internet. Conversely, a 0 indicates the absence of that intent or permission. This reflects the specific functionalities and access needs of a particular app. By analyzing these patterns, machine learning algorithms can effectively identify inconsistencies or suspicious behavior.

The dynamic dataset includes 871 API calls columns and 77 columns detailing network flows. In the CICInvesAndMal2019 dataset, the API calls columns are 2-gram API calls patterns. The 2-gram API calls refer to a method used for analyzing the sequential relationships between pairs of API calls made by Android applications. This approach is borrowed from computational linguistics, where an n-gram is a contiguous sequence of n items from a given sample of text or speech. In this approach, they obtained a dictionary of all 2-gram API calls. Each key in a dictionary represents a specific 2-gram API call pattern. For each sample, the count of how many times each unique 2-gram pattern appears is calculated. These counts serve as features for the dataset. For instance, if the 2-gram pattern appears 10 times in a sample, the count for that pattern is 10. For the network flows columns, they were extracted using the CICFlowMeter. It is used to generate bidirectional flows, allowing for separate calculations on traffic moving from source to destination and vice versa. The extracted data encompasses over 77 features. These features include fundamental aspects like duration, the volume of data packets and bytes transferred, and average packet size. Additionally, time-based features delve deeper, capturing statistics like average time between packets, minimum and maximum inter-packet delays, and periods of flow activity and inactivity (Lashkari et al., 2017).

3.3. Feature Selection

For static analysis, all features available in the CICInvesAndMal2019 dataset were used. The decision to use all features was based on the observation that each application might have unique permissions and intents, resulting in a large number of columns. Permissions and intents are critical indicators of an application's behavior and potential security risks. Permissions represent access rights that apps request to perform specific actions on the device, such as accessing the internet or reading contacts, while intents facilitate communication between different parts of an app or between apps. This approach aimed to capture all potential patterns and behaviors that could be indicative of malware. Implementing feature selection could potentially exclude some unique permissions or intents specific to certain apps, which might be crucial for classifying types of malwares.

Similarly, for dynamic analysis, all available features were used. The dynamic dataset includes features derived from API calls and network flows, which are essential for identifying abnormal or malicious behavior patterns. API calls represent the sequence of operations performed by the application, and the dataset includes 2-gram API call patterns to capture the sequential relationships between pairs of API calls. Network flows represent various aspects of network traffic, including duration, the volume of data packets, bytes transferred, and average packet size. These features are critical for detecting network-based anomalies and malicious activities.

3.4. Data Preprocessing

After data analysis, both the static and dynamic datasets will go through data preprocessing. The malware in the dataframe is an object type for both static and dynamic dataset. Thus, this malware has four types which are adware as 0, scareware as 1, SMS malware as 2, ransomware as 3 for both static and dynamic datasets as shown by two figures below. Additionally, benign is assigned as label 4 in the static dataset. This process is done using the mapping function in Python. After relabeling, four irrelevant columns are dropped from the dataframe which are the MD5, category, family, and binary type. Meanwhile for the dynamic dataset, the id and category columns are dropped from the dataframe. Both static and dynamic datasets are further divided into training 80% and testing 20% sets using the

train-test split function. Following data splitting, feature scaling is applied on the training sets of both static and dynamic datasets to standardize the features, ensuring mean is zero and standard deviation is one.

	<actionandro...	<actionandro...	<actionandro...	<actionandro...	<actionandro...	...	<category>	<MD5>	Binary_Type	new category	malware_type...
0	0	0	0	0	0	...	Adware	1c4e357a8ec...	Malware	Adware	0
1	0	0	0	0	0	...	Adware	1d15765ffee...	Malware	Adware	0
2	0	0	0	0	0	...	Adware	31c657bf77e...	Malware	Adware	0
3	0	0	0	0	0	...	Adware	37b993b5f59...	Malware	Adware	0
4	0	0	0	0	0	...	Adware	3e30f2644a2...	Malware	Adware	0
...	...	...	...	...	...	...	...	...	...	...	...
1589	0	0	0	0	0	...	SMS	0e3294e4ff1...	Malware	SMS malware	2
1590	0	0	0	0	0	...	SMS	af1863dd2ba...	Malware	SMS malware	2
1591	0	0	0	0	0	...	SMS	b1ae0d9a279...	Malware	SMS malware	2
1592	0	0	0	0	0	...	SMS	c9e3af6a442...	Malware	SMS malware	2
1593	0	0	0	0	0	...	SMS	e9068f11699...	Malware	SMS malware	2

[1594 rows x 8117 columns]

Fig. 5: Static Dataset with Mapping

	id	2Grammed_APICalls_#0	2Grammed_APICalls_#1	2Grammed_APICalls_#2	2Grammed_APICalls_#3	...	Idle_Std	Idle_Max	Idle_Min	<Category>	malware_type_encoded
0	1	0	42	0	0	...	19488345	186397186	155416258	ransomware	3
1	2	2	0	0	0	...	8283170	173387076	159763065	ransomware	3
2	3	0	40	0	0	...	22019770	263938727	230197339	ransomware	3
3	4	0	43	0	0	...	31463640	242915705	194688432	ransomware	3
4	5	2	43	0	0	...	20186565	218041946	185179596	ransomware	3
...	...	...	...	...	...	...	...	...	...	...	...
322	112	0	2	0	0	...	9253128	49854310	34664042	smsMalware	2
323	113	0	0	0	0	...	6345981	35835129	25532067	smsMalware	2
324	114	0	0	0	0	...	2404418	36331095	32462336	smsMalware	2
325	115	0	0	0	0	...	3018065	39871270	34911092	smsMalware	2
326	116	0	0	0	0	...	22179803	241166832	206910986	smsMalware	2

327 rows × 921 columns

Fig. 6: Dynamic Dataset with Mapping

3.5. Machine Learning

3.5.1. Random Forest

Random forest algorithm is an ensemble machine learning algorithm that is used for multiclass problems. The random forest models were configured with default parameters, specifically setting the number of decision trees to 100. The Gini index, the default criterion, was utilized to measure the quality of the split. It helps the random forest model pick the split that makes the groups as pure as possible, aiming to make more accurate predictions. Both random forest models for static analysis and dynamic analysis have the same parameters.

3.5.2. Artificial Neural Network

Artificial neural network is a deep learning model. After data preprocessing and removal of identifiers, the ANN model is built following the dataset. For static analysis using static dataset, the input layer of ANN will consist of 8111 nodes corresponding to the 8111 features in the static dataset. There will be five nodes in the output layer corresponding to the benign and four malware classes to be classified into. For dynamic analysis using the dynamic dataset, the input layer consists of 918 nodes and output layer consists of four nodes corresponding to the four malware classes.

Static ANN model consists of five middle layers and dynamic ANN model consists of four middle layers, as static analysis has five classes and dynamic analysis has four classes, respectively. Both static and dynamic ANN models also consist of an input layer and an output layer. Additionally, fast activation function ReLU is used for all layers except the output layer which uses softmax activation function as softmax is an output activation function specifically for multiclass problem. Finally, to accommodate the multiclass problem, the sparse categorical cross-entropy error function is chosen, which used to handle multiclass problem effectively. The ANN models are trained using 32 batch size and 20 epochs. The table 2 shows the parameters employed for the ANN models in both static and dynamic analyses. After conducting experiments with batch sizes of 32 and epochs of 20, it was found that the hidden layer size of 200 among the options of 10, 100, 200, and 500, demonstrated the best performance with the lowest loss.

Table 2: The parameters of ANN models

	ANN (Static)	ANN (Dynamic)
Hidden layers	5	4
Hidden layer size	200	200
Hidden layer activation function	ReLU	ReLU
Output layer activation function	softmax	softmax
Batch size	32	32
Epoch	20	20

3.5.3. Convolutional Neural Network

Similar to ANN, the convolutional neural network (CNN) model is a deep learning model. The CNN model is constructed to suit the characteristics of the dataset. For static analysis using the static dataset, the CNN architecture is tailored with an input layer comprising of 8111 nodes using reshape function, aligning with the 8111 features in the static dataset. The output layer is configured with five nodes, each corresponding to benign and the four malware classes for classification. For dynamic analysis using the dynamic dataset, the CNN model is adapted with an input layer featuring 918 nodes, corresponding to the 918 features in the dynamic dataset. The output layer is adjusted to incorporate four nodes, each representing one of the four malware classes.

Both static and dynamic CNN model comprise eight layers, which are an input layer, two Conv1D layers with thirty two filters, kernel size of four and ReLU activation function for features extraction, two MaxPooling1D layers with pool size of two for down-sampling, a flatten layer to convert the data into one dimensional array for feeding the next layer, a dense layer with fast activation function ReLU, and a final dense layer serves as the output layer that uses softmax activation function which is an output activation function specifically for multiclass problem. The CNN model is designed based on and referenced from Yerima & Alzaylaee (2020). They adjusted the CNN model with various parameters and evaluated them to arrive at this optimized design. Furthermore, the model employs the sparse categorical cross-entropy loss function to effectively handle the multiclass classification problem. The CNN models are trained using 32 batch size and 20 epochs. Table 3 and table 4 show the parameters of CNN models in static and dynamic analyses.

Table 3: CNN parameters for static analysis

Layer	Layer Type	Filters	Kernel Size	Pool Size	Activation Function
1	Input Layer	-	-	-	
2	Convolutional Layer	32	4	-	ReLU
3	Maximum Pooling Layer	-	-	2	-
4	Convolutional Layer	32	4	-	ReLU
5	Maximum Pooling Layer	-	-	2	-
6	Flatten Layer	-	-	-	-
7	Dense Layer	8	-	-	ReLU
8	Output Layer	5	-	-	softmax

Table 4: CNN parameters for dynamic analysis

Layer	Layer Type	Filters	Kernel Size	Pool Size	Activation Function
1	Input Layer	-	-	-	
2	Convolutional Layer	32	4	-	ReLU
3	Maximum Pooling Layer	-	-	2	-
4	Convolutional Layer	32	4	-	ReLU
5	Maximum Pooling Layer	-	-	2	-
6	Flatten Layer	-	-	-	-
7	Dense Layer	8	-	-	ReLU
8	Output Layer	4	-	-	softmax

3.6. Evaluation Metrics

To comprehensively evaluate the performance of the random forest, artificial neural network, and convolutional neural network models in both static and dynamic analysis approaches for multiclass Android malware classification, a set of performance with weighted metrics is employed. The training time and testing time were also evaluated.

Accuracy, recall, precision, and F1 score will be calculated for each model. Accuracy measures the overall proportion of correctly classified samples for all classes out of total number of samples. However, to account for multiclass in the dataset, weighted metrics will be utilized for recall, precision, and F1 score. Firstly, the recall, precision and F1 score will be calculated using the equations below for every class individually. In the equations, TP means True Positive, TN is True Negative, FP is False Positive and FN is False Negative. Then, a weight will be assigned to each class based on its representation in the dataset. Finally, the weighted metrics will be computed by multiplying the individual class score by its corresponding weight and summing these products across all classes.

$$Multiclass\ Accuracy = \frac{Correct\ classification\ for\ all\ classes}{Total\ number\ of\ samples}$$

$$Precision\ = \frac{TP}{(TP\ +\ FP)}$$

$$Recall\ = \frac{TP}{(TP\ +\ FN)}$$

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

$$w_i = \frac{No.of\ samples\ in\ class\ i}{Total\ number\ of\ samples}$$

$$Multiclass\ Precision\ = \sum_{i=1}^{5} w_i \times Precision_i$$

$$Multiclass\ Recall = \sum_{i=1}^{5} w_i \times Recall_i$$

$$Multiclass\ F1\ score = \sum_{i=1}^{5} w_i \times F1\ score_i$$

4. Results

Table 5 shows the performance of models using static analysis. Three models were evaluated, which are random forest, artificial neural network, and convolutional neural network. The metrics used for evaluation include accuracy, recall, precision, and F1 score. Random forest exhibits the highest accuracy at 95.30%, followed by the artificial neural network at 94.36% and the convolutional neural network at 92.48%. Random forest also achieves better results in recall, precision, and F1 score than deep learning models. Figure 7 below shows the confusion matrices for each model in static analysis.

Table 5: Models Performance using Static Analysis Table (best results in bold)

Model	Accuracy (%)	Recall (%)	Precision (%)	F1 (%)
Random Forest	**95.30**	**95.30**	**95.46**	**94.98**
Artificial Neural Network	94.36	94.36	94.26	94.11
Convolutional Neural Network	92.48	92.48	93.23	92.61

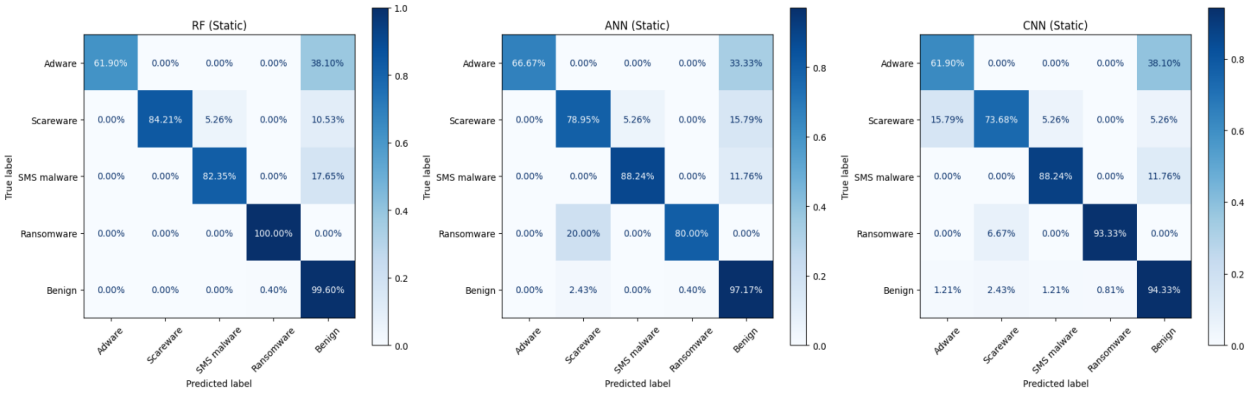

Fig. 7: Confusion Matrices for All Models in Static Analysis

Table 6 shows the training and testing times of the models when employing static analysis. Notably, random forest requires the shortest training time of merely 2.26 seconds, highlighting its efficiency in model training. Artificial neural network takes more time at 47.43 seconds, and convolutional neural network takes the longest at 203.69 seconds, indicating the computational intensity associated with its training. Random forest also takes the least time for testing compared to deep learning models. The results highlight that random forest is faster and performs better than deep learning models.

Table 6: Models Training Time using Static Analysis (best results in bold)

Model	Training Time (s)	Testing Time (s)
Random Forest	**2.26**	**0.02**
Artificial Neural Network	47.43	0.34
Convolutional Neural Network	203.69	0.85

Shifting focus to dynamic analysis, Table 7 delves into the performance of models under this approach. The results show a notable decrease in performance across all metrics compared to static analysis. Random forest still maintains the highest accuracy at 78.79%, followed by the convolutional neural network and the artificial neural network with the same accuracy 71.21%. Random forest also achieves the best result in recall, precision and F1 score. Figure 8 shows the confusion matrices for each model in dynamic analysis.

Table 7: Models Performance using Dynamic Analysis (best results in bold)

Model	Accuracy (%)	Recall (%)	Precision (%)	F1 (%)
Random Forest	**78.79**	**78.79**	**81.13**	**78.72**
Artificial Neural Network	71.21	72.03	77.88	73.99
Convolutional Neural Network	71.21	71.21	74.12	71.51

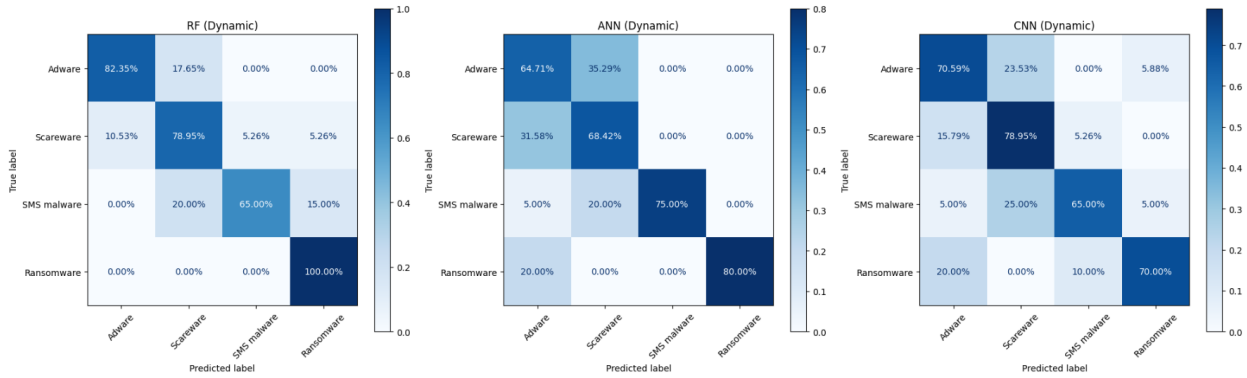

Fig. 8: Confusion Matrices for All Models in Dynamic Analysis

Table 8 examines the training and testing times of models under dynamic analysis. Interestingly, the training times for dynamic analysis are considerably shorter compared to static analysis due to the smaller dataset for dynamic analysis. Random forest leads with the shortest training time of 0.36 seconds, followed by artificial neural network at 3.97 seconds, and convolutional neural network at 11.46 seconds. Despite the shorter training and testing times observed in dynamic analysis compared to static analysis, the relative positions of the models remain consistent. Random forest still stands as the fastest model, followed by the artificial neural network and the convolutional neural network. This reaffirms the efficiency and performance of random forest in both static and dynamic analysis compared to deep learning models.

Table 8: Models Training Time using Static Analysis (best results in bold)

Model	Training Time (s)	Testing Time (s)
Random Forest	**0.36**	**0.01**
Artificial Neural Network	3.97	0.27
Convolutional Neural Network	11.46	0.29

Figure 9 provides a comprehensive comparison of the performance metrics, including accuracy, precision, recall, and F1-score, for random forest, artificial neural network, and convolutional neural network in both static and dynamic analyses. Figure 10 shows the comparative analysis of the training and testing times across random forest, artificial neural network, and convolutional neural network in both static and dynamic analyses.

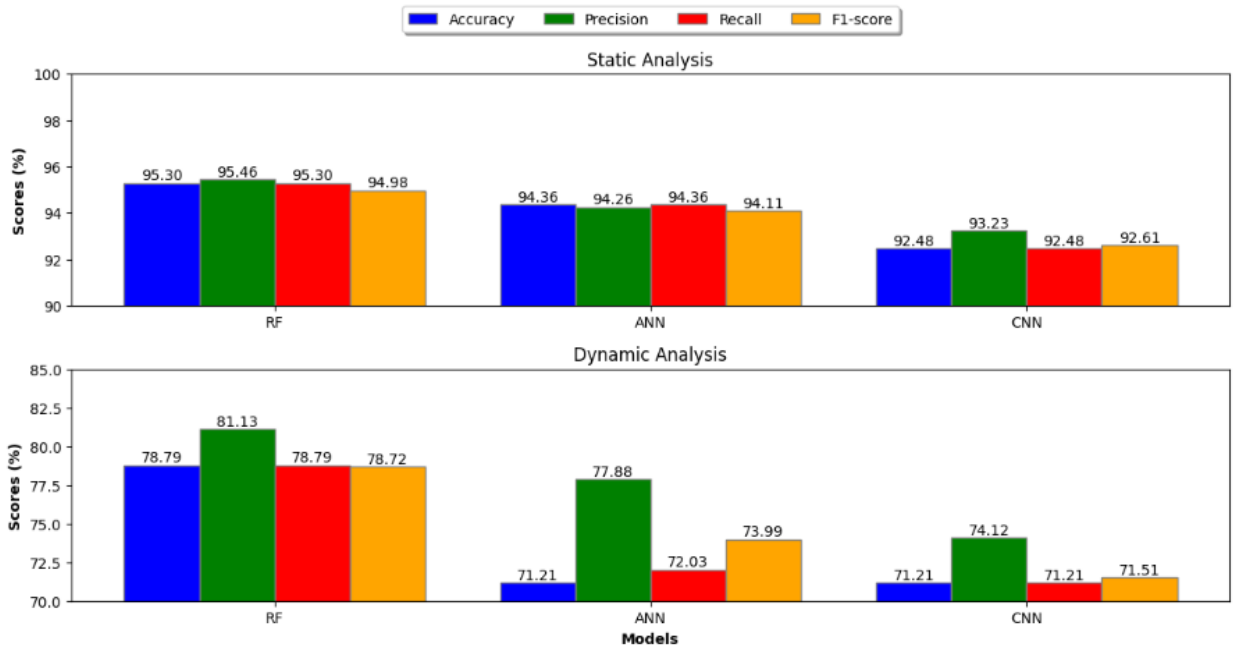

Fig. 9: Performance Metrics Comparison of Machine Learning Models

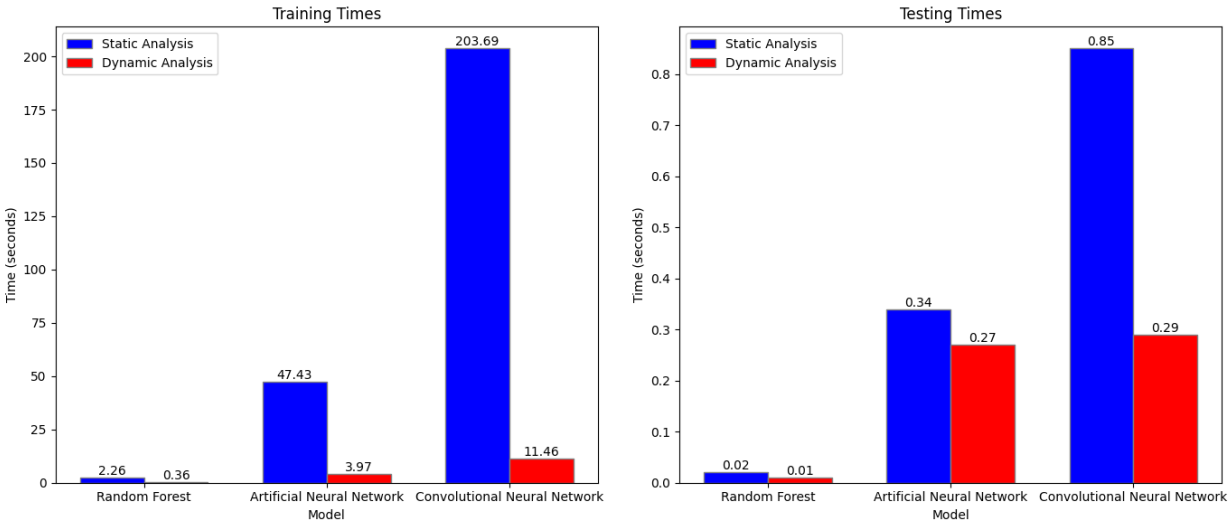

Fig. 10: Training and Testing Times Comparison of Machine Learning Models

The results revealed a difference in performance between random forest and deep learning models across both static and dynamic analyses. Random forest consistently achieved superior accuracy, recall, precision, and F1 score compared to deep learning models. The observed superiority of random forest over deep learning models in both static and dynamic analyses may be attributed to factors such as the small and imbalanced nature of the dataset, potentially leading to overfitting in deep learning models. These results suggest that for smaller and imbalanced datasets, random forest models may be more reliable for malware classification in cybersecurity systems.

Moreover, the results showed that static analysis methods outperformed dynamic analysis techniques, which aligns with the findings reported in the paper by Bayazit et al. (2023) using the same dataset. As noted in their work, the dynamic features, particularly the network traffic data, do not exhibit a normal distribution and encompass fewer features compared to the static analysis. This characteristic of the dynamic data could contribute to the lower accuracy observed in the dynamic analysis methods. Considering these insights, the superior performance of static analysis in the current study could be attributed to the more comprehensive and well distributed static features, which provide a better representation for effective malware classification.

5. Discussion

The results revealed the difference in performance between random forest and deep learning models in both static and dynamic analysis of Android OS malware. While random forest consistently achieved superior accuracy, recall, precision, and F1 score, deep learning models achieved inferior results in both analysis types.

The observed superiority of random forest over deep learning models in both static and dynamic analyses can be attributed to various factors. Firstly, the parameter optimization plays a crucial role. In the study from Shar et al. (2020), they showed that machine learning classifiers performed better than deep learning classifiers in static analysis for malware detection. They indicated that one of the reasons lies in the complexity of parameter tuning for deep learning models. Unlike machine learning classifiers, deep learning models often require complicated parameter optimization to be optimal (Shar et al., 2020).

Another factor to consider is the potential for overfitting in deep learning models, especially when dealing with imbalanced and small datasets for Android malware classification. In the static dataset, the malware classes have less samples compared to benign samples. Moreover, both static and dynamic datasets have limited samples. Deep learning models can be susceptible to overfitting on the limited malware data, leading to a decrease in generalizability on unseen data (Tayyab et al., 2022). Also mentioned in Wang et al. (2023), the performance of deep learning models is greatly affected by overfitting on imbalanced small dataset. They proposed using a modern synergetic neural network that

can effectively handle imbalanced sample sizes and improve classification performance. Modern synergetic neural network allows its parameters to adjust automatically, which helps the network to adapt to imbalanced and small datasets. Random forest, on the other hand, is generally less prone to overfitting due to its inherent ensemble nature. This characteristic might be advantageous in the context of malware multiclass classification, where robust performance on unseen malicious samples is critical.

From the experiments, random forest outperforms deep learning models in both static and dynamic analysis of Android OS malware classification. This advantage stems from random forest's less complex architecture and inherent resistance to overfitting, especially beneficial with the imbalanced and potentially limited datasets common in malware classification. Consequently, random forest offers a faster and more effective approach for these specific tasks.

In the investigation conducted by Bayazit et al. (2023) which uses the same dataset, they focused exclusively on binary classification, distinguishing between malware and benign software, this research extends to multiclass classification. Notably, they reported that their LSTM model achieved the highest accuracy in static analysis, outperforming random forest. While in dynamic analysis, their CNN-LSTM model has superior results. Conversely, in this research with multiclass classification approach, the random forest is the best model in both static and dynamic analyses.

The study's exploration of multiclass classification and dynamic analysis in Android malware detection will bring enhancement to the existing security frameworks. By specifically targeting different types of malwares and employing real-time behavioral analysis, the proposed approach offers a strategy for improvements in the accuracy and time efficiency of detection systems. Integrating multiclass classification into security systems helps organizations to not only identify malware more accurately but also respond more effectively to specific threat scenarios.

Notably, while this investigation provides valuable insights into Android malware classification, it is constrained by the use of a single dataset, which may limit the generalizability of the results. Additionally, the study focuses on a limited set of machine learning algorithms, overlooking other promising approaches.

6. Conclusion

This study compares the performance of random forest, artificial neural network, and convolutional neural network in detecting and classifying Android malware using static and dynamic analysis methods. The results demonstrate that random forest outperforms deep learning models in both accuracy and efficiency, making it a promising candidate for resource-constrained environments. Static analysis, which examines permissions and intents, proves to be more effective than dynamic analysis based on API calls and network flows. These findings underscore the importance of selecting appropriate machine learning algorithms and analysis methods for Android malware detection.

The study contributes to the growing body of research on Android malware detection by addressing the gaps in multiclass classification and dynamic analysis. The insights gained from this study can inform the development of more robust and efficient Android malware detection systems that can adapt to the evolving landscape of mobile threats. However, further research is needed to validate the findings on diverse datasets and explore the potential of hybrid analysis techniques that combine the strengths of static and dynamic approaches. Modern synergetic neural network proposed by Wang et al. (2023) could also be explored in future research to overcome the overfitting issues in deep learning models caused by imbalanced small dataset. To address the limitations of the use of single dataset and limited set of machine learning algorithms, future research could explore the use of different datasets from multiple sources to validate the findings across different contexts. Moreover, a wider range of machine learning algorithms like ensemble methods and deep learning architectures could offer a better comparison and identification of the most effective techniques for Android malware classification.

As the sophistication and prevalence of Android malware continue to rise, it is crucial for

researchers and practitioners to collaborate and share knowledge to stay ahead of the curve. This study serves as a foundation for future work in this field, highlighting the need for continuous innovation in machine learning based Android malware detection. By building on the findings of this study and addressing its limitations, the academia and industry communities can contribute to the development of more secure and trustworthy mobile ecosystems.

References

A, S., S, S. R., G, V., Selvan, G. S. R. E., & Ramkumar, M. P. (2023). Malware Detection in Android Application using Static Permission. *2023 5th International Conference on Inventive Research in Computing Applications (ICIRCA)*, 1241–1245. https://doi.org/10.1109/ICIRCA57980.2023.10220934

Almahmoud, M., Alzu'bi, D., & Yaseen, Q. (2021). Redroiddet: Android malware detection based on recurrent neural network. *Procedia Computer Science*, *184*, 841–846. https://doi.org/10.1016/j.procs.2021.03.105

Amenova, S., Turan, C., & Zharkynbek, D. (2022). Android Malware Classification by CNN-LSTM. *SIST 2022 - 2022 International Conference on Smart Information Systems and Technologies, Proceedings*. https://doi.org/10.1109/SIST54437.2022.9945816

Anuar, N. A., Mas'ud, M. Z., Bahaman, N., & Ariff, N. A. M. (2020). Analysis of Machine Learning Classifier in Android Malware Detection through Opcode. *2020 IEEE Conference on Application, Information and Network Security, AINS 2020*, 7–11. https://doi.org/10.1109/AINS50155.2020.9315060

Bayazit, E. C., Sahingoz, O. K., & Dogan, B. (2022). A Deep Learning Based Android Malware Detection System with Static Analysis. *HORA 2022 - 4th International Congress on Human-Computer Interaction, Optimization and Robotic Applications, Proceedings*. https://doi.org/10.1109/HORA55278.2022.9800057

Bayazit, E. C., Sahingoz, O. K., & Dogan, B. (2023). Deep Learning based Malware Detection for Android Systems: A Comparative Analysis. *Tehnicki Vjesnik*, *30*(3), 787–796. https://doi.org/10.17559/TV-20220907113227

Elayan, O. N., & Mustafa, A. M. (2021). Android malware detection using deep learning. *Procedia Computer Science*, *184*, 847–852. https://doi.org/10.1016/j.procs.2021.03.106

Eltaher, A., Abu-Juma'a, D., Hashem, D., & Alawneh, H. (2023). Design and Implementation of a Malware Detection Tool Using Network Traffic Analysis in Android-based Devices. *2023 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology, JEEIT 2023*, 276–280. https://doi.org/10.1109/JEEIT58638.2023.10185826

Fiky, A. H. El, Elshenawy, A., & Madkour, M. A. (2021). Detection of Android Malware using Machine Learning. *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference, MIUCC 2021*, 9–16. https://doi.org/10.1109/MIUCC52538.2021.9447661

Hadiprakoso, R. B., Kabetta, H., & Buana, I. K. S. (2020). Hybrid-Based Malware Analysis for Effective and Efficiency Android Malware Detection. *Proceedings - 2nd International Conference on Informatics, Multimedia, Cyber, and Information System, ICIMCIS 2020*, 8–12. https://doi.org/10.1109/ICIMCIS51567.2020.9354315

Haidros Rahima Manzil, H., & Manohar Naik, S. (2022). DynaMalDroid: Dynamic Analysis-Based Detection Framework for Android Malware Using Machine Learning Techniques. *IEEE International Conference on Knowledge Engineering and Communication Systems, ICKES 2022*. https://doi.org/10.1109/ICKECS56523.2022.10060106

Haidros Rahima Manzil, H., & Manohar Naik, S. (2024). Detection approaches for android malware: Taxonomy and review analysis. In *Expert Systems with Applications* (Vol. 238). Elsevier Ltd. https://doi.org/10.1016/j.eswa.2023.122255

HR, S. (2019). Static Analysis of Android Malware Detection using Deep Learning. *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, 841–845. https://doi.org/10.1109/ICCS45141.2019.9065765

Ibrahim, M., Issa, B., & Jasser, M. B. (2022). A Method for Automatic Android Malware Detection Based on Static Analysis and Deep Learning. *IEEE Access*, *10*, 117334–117352. https://doi.org/10.1109/ACCESS.2022.3219047

Imtiaz, S. I., Rehman, S. ur, Javed, A. R., Jalil, Z., Liu, X., & Alnumay, W. S. (2021). DeepAMD: Detection and identification of Android malware using high-efficient Deep Artificial Neural Network. *Future Generation Computer Systems*, *115*, 844–856. https://doi.org/10.1016/j.future.2020.10.008

Kabakus, A. T. (2022). DroidMalwareDetector: A novel Android malware detection framework based on convolutional neural network. *Expert Systems with Applications*, *206*. https://doi.org/10.1016/j.eswa.2022.117833

Lashkari, A. H., Gil, G. D., Mamun, M. S. I., & Ghorbani, A. A. (2017). Characterization of tor traffic using time based features. *ICISSP 2017 - Proceedings of the 3rd International Conference on Information Systems Security and Privacy*, *2017-January*, 253–262. https://doi.org/10.5220/0006105602530262

Lashkari, A. H., Kadir, A. F. A., Taheri, L., & Ghorbani, A. A. (2018). Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification. *2018 International Carnahan Conference on Security Technology (ICCST)*, 1–7. https://doi.org/10.1109/CCST.2018.8585560

Mbunge, E., Muchemwa, B., Batani, J., & Mbuyisa, N. (2023). A review of deep learning models to detect malware in Android applications. *Cyber Security and Applications*, *1*, 100014. https://doi.org/10.1016/j.csa.2023.100014

Mohamad Arif, J., Ab Razak, M. F., Tuan Mat, S. R., Awang, S., Ismail, N. S. N., & Firdaus, A. (2021). Android mobile malware detection using fuzzy AHP. *Journal of Information Security and Applications*, *61*. https://doi.org/10.1016/j.jisa.2021.102929

Muzaffar, A., Ragab Hassen, H., Lones, M. A., & Zantout, H. (2022). An in-depth review of machine learning based Android malware detection. *Computers & Security*, *121*, 102833. https://doi.org/10.1016/j.cose.2022.102833

Raghuvanshi, P., & Singh, J. P. (2023). *Android Malware Detection Using Machine Learning Techniques*. 1117–1121. https://doi.org/10.1109/csci58124.2022.00200

Sarah, N. Al, Rifat, F. Y., Hossain, M. S., & Narman, H. S. (2021). An Efficient Android Malware Prediction Using Ensemble machine learning algorithms. *Procedia Computer Science*, *191*, 184–191. https://doi.org/10.1016/j.procs.2021.07.023

Selvaganapathy, S. G., Sadasivam, S., & Ravi, V. (2021). A Review on Android Malware: Attacks, Countermeasures and Challenges Ahead. *Journal of Cyber Security and Mobility*, *10*(1), 177–230. https://doi.org/10.13052/jcsm2245-1439.1017

Shar, L. K., Demissie, B. F., Ceccato, M., & Minn, W. (2020). Experimental comparison of features and classifiers for Android malware detection. *Proceedings - 2020 IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems, MOBILESoft 2020*, 50–60. https://doi.org/10.1145/3387905.3388596

Shatnawi, A. S., Yassen, Q., & Yateem, A. (2022). An Android Malware Detection Approach Based on Static Feature Analysis Using Machine Learning Algorithms. *Procedia Computer Science*, *201*(C), 653–658. https://doi.org/10.1016/j.procs.2022.03.086

Shyong, Y.-C., Jeng, T.-H., & Chen, Y.-M. (2020). Combining Static Permissions and Dynamic Packet Analysis to Improve Android Malware Detection. *2020 2nd International Conference on Computer Communication and the Internet (ICCCI)*, 75–81. https://doi.org/10.1109/ICCCI49374.2020.9145994

Taheri, L., Kadir, A. F. A., & Lashkari, A. H. (2019). Extensible Android Malware Detection and Family Classification Using Network-Flows and API-Calls. *2019 International Carnahan Conference on Security Technology (ICCST)*, 1–8. https://doi.org/10.1109/CCST.2019.8888430

Tayyab, U. E. H., Khan, F. B., Durad, M. H., Khan, A., & Lee, Y. S. (2022). A Survey of the Recent Trends in Deep Learning Based Malware Detection. In *Journal of Cybersecurity and Privacy* (Vol. 2, Issue 4, pp. 800–829). Multidisciplinary Digital Publishing Institute (MDPI). https://doi.org/10.3390/jcp2040041

Wang, Z., Li, H., & Ma, L. (2023). Modern synergetic neural network for imbalanced small data classification. *Scientific Reports*, *13*(1). https://doi.org/10.1038/s41598-023-42689-8

Wang, Z., Liu, Q., & Chi, Y. (2020). Review of android malware detection based on deep learning. In *IEEE Access* (Vol. 8, pp. 181102–181126). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2020.3028370

Yang, S., Wang, Y., Xu, H., Xu, F., & Chen, M. (2022). An Android Malware Detection and Classification Approach Based on Contrastive Lerning. *Computers and Security*, *123*. https://doi.org/10.1016/j.cose.2022.102915

Yerima, S. Y., & Alzaylaee, M. K. (2020). Mobile Botnet Detection: A Deep Learning Approach Using Convolutional Neural Networks. *2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, 1–8. https://doi.org/10.1109/CyberSA49311.2020.9139664