BarkDroid: Android Malware Detection Using Bark Frequency Cepstral Coefficients

P Tarwireyi[*1], A Terzoli[2], M O Adigun[3]

[1-3]University of Zululand

E-mail: tarwireyip@unizulu.ac.za[1], terzolia@unizulu.ac.za[2], adigunm@unizulu.ac.za[3]

Submitted: 4 August 2022, revised: 19 August 2022, accepted: 26 August 2022

Abstract. Since their inaugural releases in 2007, Google's Android and Apple's iOS have grown to dominate the mobile OS market share. Currently, they jointly possess over 99% of the global market share with Android being the leading mobile Operating System of choice worldwide, controlling close to 70% of the market share. Mobile devices have enabled the exponential growth of a plethora of mobile applications that play key roles in enabling many use cases that are pivotal in our daily lives. On the other hand, access to a large pool of potential end users is available to both legitimate and nefarious applications, thus making mobile devices a burgeoning target of malicious applications. Current malware detection solutions rely on tedious, time-consuming, knowledge-based, and manual processes to identify malware. This paper introduces BarkDroid, a novel Android malware detection technique that uses the low-level Bark Frequency Cepstral Coefficients audio features to detect malware. The initial results obtained show that Bark Frequency Cepstral Coefficients have high discriminative capabilities to achieve accurate preditions. BarkDroid achieved 97.9% accuracy, 98.5% precision, an F1 score of 98.6%, and shorter execution times.

Keywords: Android malware detection, malware classification, bark frequency cepstral coefficients.

1. Introduction

Mobile devices have become an indispensable part of our daily lives. They have caused a paradigm shift in the way people used to live, learn, communicate, collaborate, and conduct business. Despite the fact that they only started in 1973, their proliferation has been so rapid that it was estimated that the number of people using mobile devices had reached 6.3 billion by 2021. This number is projected to rise to 7.7 billion in 2027 [1].

Since their advent in the early 70's, mobile devices have taken enormous leaps and bounds to introduce feature sets and functionalities that puts them on par with, if not exceeding, handheld computers. Advances in electronics, cutting-edge technologies, and the maturity of operating system ecosystems have led to mobile devices incorporating many advanced functionalities that were previously thought to be only possible with computers. Additionally, the rapid evolution of mobile devices from the initially bulky and expensive device which could only make calls, to the present-day smartphone that has been miniaturised and is jam-packed with advanced features, has been partly due to the ever-growing use cases and user expectations [2].

At the core of the mobile device sits the operating system (OS), which acts as an intermediary between the mobile device hardware and users. Its main responsibilities include managing device hardware, and software and providing utilities and interfaces that enable users to interact with mobile devices. As far as mobile operating systems are concerned, Google's Android and Apple's iOS are the most successful operating systems [1].

Since their inaugural releases in 2007, Google's Android and Apple's iOS have grown to dominate the mobile OS market share. Currently, they jointly possess over 99% of the global market share with Android being the leading mobile Operating System of choice worldwide, controlling close to 70% of the market share [2]. Mobile devices have enabled the exponential growth of a plethora of mobile applications that play key roles in enabling many use cases which are pivotal in our daily lives. This means mobile applications have become a forte of the mobile phone ecosystem. On the other hand, mobile applications have also become the Achilles heel because the ability to access a large pool of potential end users is not only available to legitimate applications, but also to malicious ones. The term malware refers to any software that deliberately infects and causes harm to computing systems.

If we can track the historical malware patterns, it can be noted that attackers have always preferred targeting popular platforms to maximise their chances. This means that in their unprecedented attack campaigns and sophistication, hackers also play the numbers game. They cast their net as wide as possible, hoping to compromise as many as possible. Consequently, it is unsurprising that although all mobile operating systems have been attacked by malware, Android has been the lucrative primary target for malware attacks. Android is reported to host roughly 99% of known mobile malware and is the focus of most research efforts in mobile malware detection [1], [2]. It used to be the Microsoft Windows operating system on personal computers; now, it is Android. Unfortunately, the permission system meant to be the first line of defence in the Android system has been ineffective. The assumption was that users will scrutinise the permissions that an app will request at installation and only allow them when necessary. However, this was never due to the users' naivety, lack of knowledge, and unsuspecting nature [3].

Android malware can steal, corrupt, or delete user data causing stress and financial loss. According to Symantec 2019 [4], even though around 24000 malicious mobile apps are blocked daily, a sizeable number still managed to find ways to bypass detection. Malware developers have been generating new malware using techniques such as module reuse and automated generation tools. For several decades, antivirus solutions have been the defacto malware mitigation strategy. Traditionally, such solutions have primarily been reactive. They rely on known fixed string patterns or signatures to detect malware [5]. With signature-based malware detection solutions, applications are scanned while searching a database for predefined matching patterns. The sheer number of applications that have successfully bypassed such systems is testimony that this countermeasure is ineffective. Not only because it does not give zero-day insurance, but it also does not scale very well in the face of the astronomical rate of malware generation per day.

Malicious applications pose an enormous security threat to mobile devices. Current malware detection solutions generally rely on time-consuming, knowledge-based, and manual processes to identify malware. This has serious shortcomings, especially against new and unknown malware. Signature-based malware solutions are not able to detect modern malware that uses packing and smart coding techniques such as polymorphism, metamorphism, and other evasive techniques that quickly change the malware behaviours and generate a large number of new variants which are predominantly variants of existing malware.

As we more and more rely on mobile devices that have become hosts for our sensitive data and applications, there is a need to develop new intelligent malware detection systems that besides detecting known attacks also have the ability to provide zero-day insurance. Such systems should be able to cope with the scale and complexity of malware applications being generated every day.

2. Literature review

Several previous works have investigated different android malware detection techniques [5]. Static and dynamic analysis techniques have been used in literature to ascertain whether an android app is malicious or not [6].

2.1. Static analysis

Several static features have been proposed in literature. These include:

- Requested permissions – is the first line of defence provided by the Android operating system to restrict access to data and actions that the app can perform [6]–[10]. Permissions are a major source of malware infection [11], [12] [13], [14]. Studies that use permissions for malware detection generate attribute feature vectors from the AndroidManifest.xml file where a one is assigned if the permission is present; otherwise, a zero is assigned [15], [16][17]–[19]. Other studies use text classification techniques such as Term Frequency-Inverse Document Frequency [6]. Researchers have noted that malicious applications tend to request many dangerous permissions [15], [18].
- Hardware components – access to the hardware is explicitly declared in the manifest file. Some hardware components are red flag signs, for instance, GPS, mic, and Internet should be viewed with scrutiny [20], [21]
- API calls and Intents – unnecessary access to sensitive resources can be a sign of malevolent intentions, for example getDeviceId() – IMEI [12], [14] [21], [22].
- Opcode sequences - Android applications are generally developed in Java and then compiled and converted to the optimised Dalvik bytecode, an executable format for Android applications. Dalvik bytecode of compiled applications can be used to distinguish malware from benign applications. Most works have disassembled the Dex to extract the opcodes and then use the n-grams of opcodes in machine learning analysis [22]–[24].

2.2. Dynamic analysis

Dynamic malware analysis techniques that have been used in literature include:

- Resource utilisation – Resource usage is monitored whilst the installed app is being tested. This includes CPU, memory, network usage, API calls, and energy consumption [25], [26].
- System calls – researchers have used agent-based systems to collect system calls and generate unique signatures or text sequences that can be used for malware detection [27], [28].

While there exists some prior work that utilises static features such as acoustic signals and images [4], [29], [30] to analyse and detect malware, there remain a number of issues that should be explored in future research [31]. This work is still very limited quantity-wise and in its infancy stage, but it should be noted that no negative result have been reported up to now, to suggest that there is no value in further exploration. As an illustration of current limitations related to this use of static features, the researchers that have looked at using APKs to detect malware, in most instances, have only used the dex file for their analysis. This neglects the other files that offer opportunities for generating more discriminative features that can be used to improve overall detection accuracy. Moreover, they only utilise the typical audio features, Mel-frequency cepstral coefficients (MFCC), which represent the short-term power spectrum of the generated malware audio signal [31].

Evidence from recent studies suggests that machine learning can be used as a viable solution for malware classification [33], [34]. Android malware detection using machine learning is a complex task due to the ever-changing malware evasion techniques and the lack of well-defined features which can be used to distinguish the various android applications with high fidelity [29]. Despite there being a body of work that has been used to find the best way of classifying android malware, it remains unknown which feature set is the best. It is generally agreed that malware detection is an undecidable problem which warrants ongoing research efforts.

3. Research Method

The raw Android Application Package is an archive that contains various components of the android application and is not suitable to be directly fed to machine learning algorithms for automatic analysis and prediction. Each Android Application Package carries components such as libraries, methods, classes, certificates, assets, resources, and configuration files that make it functional. The variances in the content, nature, and compositions of these components make the unique characteristics of each android application. Selecting appropriate distinguishable characteristics from the raw Android Application Package is an open area of research. In order to improve accuracy and reduce the number of false positives in android malware detection, strategies are needed for discarding irrelevant details and only selecting relevant information. Such relevant features should possess stable and effective discriminative characteristics that will enable machine learning algorithms to distinguish between various types of android applications.

The research reported in this study is inspired by similar work in the audio engineering field [35], [36]. Our work treats an android application as a signal modeled so that it carries unique characteristics that can be analysed using Automated Signal Recognition techniques. At the low level, an android application is simply translated into a series of ones and zeros. This has similar characteristics to signals such as audio which is a sequence of sounds translated to a series of ones and zeros to represent the oscillating longitudinal waves. Such waveform representation renders itself nicely for automatic analysis and processing in digital systems.

From a high-level point of view, this study employed a simple two-phase machine learning life cycle that uses data engineering as the first phase, followed by model engineering. Data engineering is concerned with building systems and processes for raw data ingestion, storage, wrangling, feature creation, and transforming into formats useful for analysis. On the other hand, model engineering is an iterative process of writing, executing, and tuning machine learning models. A graphical description of the procedure is presented in Figure 1.

- *Step 1:* Dataset is collected and unzipped into respective folders
- *Step 2:* The dataset is cleaned to de-duplicate and remove corrupted APK files
- *Step 3:* Data exploration and validation are performed to ascertain the distribution of the different samples contained in the dataset
- *Step 4:* Using algorithm 1 given below, traverse through all the folders and subfolders in the dataset to convert APK files into WAV files. The output of this step is a novel malware audio dataset that can be used by the research community to carry out further analysis.
- *Step 5:* Like the raw APK file that is not suitable for feeding directly as input into automatic recognition systems, wav files are also unsuitable. There is a need for an intermediary step that will extract relevant information analysable by acoustic models. Because the generated audio signals exhibited characteristics that are similar to those seen in noisy signals, such as the

Watkins Marine Mammal Sound Database, there is a need to consider features that exhibit superior noise robustness. This study uses the bark-frequency cepstrum coefficients algorithm highlighted in figure 2 to extract features. This algorithm segments the waveform and uses a combination of low and high pass filters to bark frequency cepstrum.

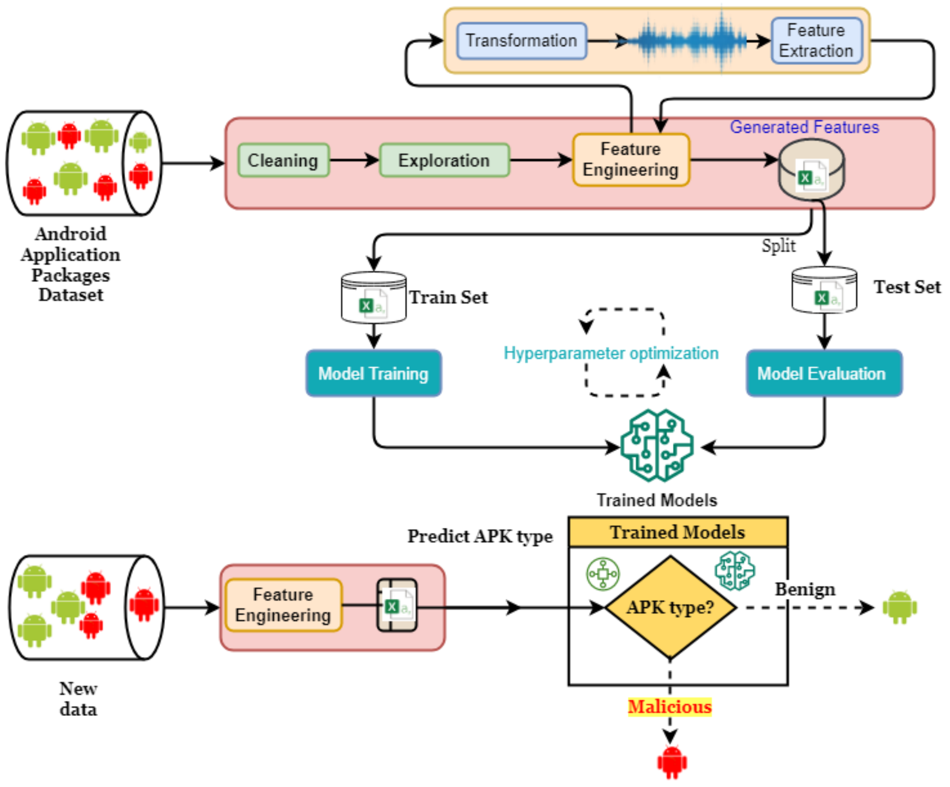

Figure 1. Proposed methodology architecture diagram

Bark Frequency Cepstrum is the short-time power spectrum representation of a signal based on the linear cosine transform of a log spectrum on a non-linear Bark scale of frequency [35], [36]. Figure 2 shows the block diagram for the bark-frequency cepstrum coefficients algorithm. The bark frequency is calculated as shown in equations (1) and (2) [35].

$$f_{bark} = 6 * sinh^{-1}(\frac{f}{600}) \quad (1)$$

$$f = 600 * sinh(\frac{f_{bark}}{6}) \quad (2)$$

Where f is the waveform's linear frequency in hertz and f_{bark} is the resultant frequency in bark.

```
Algorithm 1:  Transforming APK to Audio
Input:        APK Dataset directory: $D_{APK}$ = { $d_{apk1}$, $d_{apk2}$, …, $d_{apkn}$)
Result:       WAV Dataset directory: $D_{WAV}$ = { $d_{wav1}$, $d_{wav2}$, …, $d_{wavn}$}

For all $d_{apki} \in D_{APK}$ do:            {Where $d_{apki}$ includes all level sub folders}
        For all apks $\in d_{apki}$ do:
                Read apk into memory
                Binarize APK (Covert into a series of ones and zeros)
                Put the bits into 16bit groups
                Write the resultant oscillating longitudinal wave to a .wav file in $d_{wavi}$
        End for
End for
```

Pre-emphasis is applied to the audio signal as a filter to compensate for the average spectral shape. Windowing is used to split the input signal into short enough temporal segments, which do not allow enough time for the properties of the signal to change in each segment [35]. To determine the perceived loudness of frequencies at given sound pressure levels, the outputs of the bark scale filter banks are weighted according to the Fletcher Munson or equal loudness curve. The signal is compressed using the logarithmic function and passed through the discrete cosine transform, a time-frequency transform operation for decorrelating sequentially correlated data [36].

- *Step 6:* The resultant dataset of extracted features is split into training, validation, and testing sets.
- *Step 7:* Models are created, trained, and validated to learn the intrinsic patterns that can be used to distinguish between malicious and benign android application package files. The train and validation phases are iteratively repeated until optimal performance levels have been obtained. This phase includes hyperparameter optimisation.
- *Step 8:* The generalisability of the trained classifier is tested when it is used to make predictions on data it has never seen before. This data comes from the testing set.

3.1. Datasets

We use the CICMalAnal2017 [32] and CICMalDroid 2020 [33] [34] datasets provided by the Canadian Institute for Cybersecurity. The CICMalDroid 2020 dataset consists of 17341 android application packages collected from several sources such as Contagio blog, AMD, Maldozer, and other recent and sophisticated datasets collected until 2018. The dataset covers five broad categories of android malware. Details of the datasets are given in tables 1 and 2.

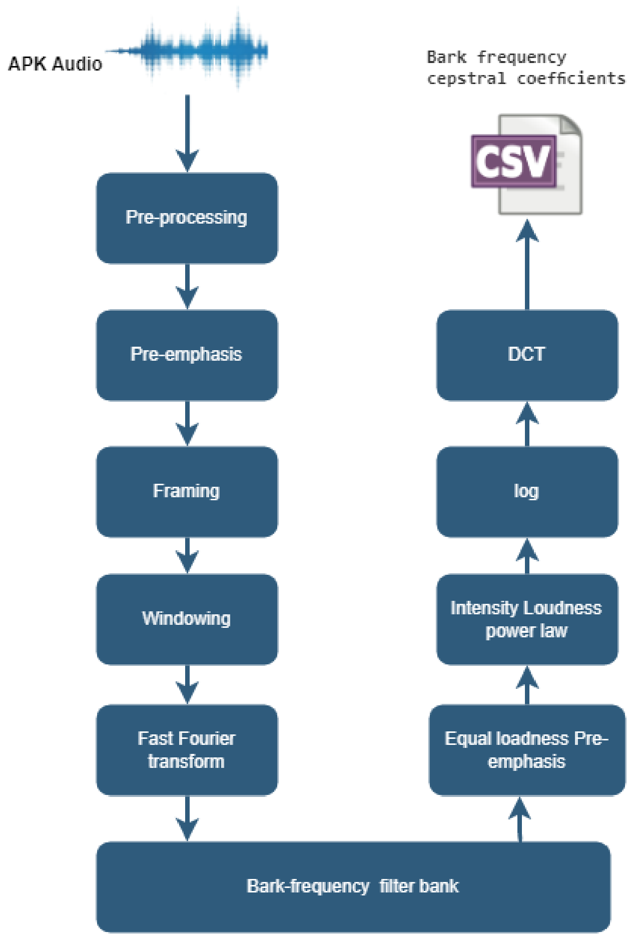
APK Audio
Bark frequency cepstral coefficients
CSV
Pre-processing
Pre-emphasis
Framing
Windowing
Fast Fourier transform
Bark-frequency filter bank
Equal loadness Pre-emphasis
Intensity Loudness power law
log
DCT

Figure 2. Bark-Frequency Cepstrum Coefficients algorithm block diagram [36]

Table 1. CICMalAnal2017 dataset.

Category	**Description**	**Number of samples**	**Number of resultant audio samples**
Adware	This unwanted program works by repeatedly displaying pop-up adverts on the mobile screen to generate revenue for its authors.	104	104
Ransomware	The malware encrypts the victims' files and demands a ransom to restore access.	101	101
Scareware	Is a form of malware that uses social engineering to manipulate victims into buying malicious software.	112	112
SMSmalware	This malware utilises short messaging services and other mobile messaging services to exploit mobile devices. It sends malicious SMSes and intercepts	109	109

	SMSes to steal passwords, online banking details, and other information from infected devices.		
Benign	These are legitimate applications that were scanned with VirusTotal to certify that they are not malicious.	1 700	1 648
TOTAL		**2 126**	**2 074**

Table 2. CICMalDroid2020 dataset.

Category	**Description**	**Number of samples**	**Number of resultant audio samples**
Adware	Same as in table 1.	1 515	1 514
Banking Malware	Infiltrate mobile devices to attempt to convince users to divulge sensitive banking details such as credit card number, login username, password, or pin. In most cases, the apps are designed as a trojan that will send the exploited data to the cybercriminal.	2 506	2 504
SMS Malware	As provided in the previous table	4 822	4 821
Mobile Riskware	These are legitimate applications that are potentially risky because they leave users and systems vulnerable to legal risks and data and application exploits. For example, they might expose vulnerabilities that cybercriminals might exploit to access the kernel and misuse programs to exfiltrate data.	4 362	3 904
Benign	As provided in the previous table	4 042	4 039
TOTAL		**17 247**	**16 783**

4. Result and Discussion

In this section, we present the experimental results of the proposed method. It should be noted that the discussion will be preliminary and focused on assessing how promising our method is. A more detailed interpretation of every single result will be the focus of future papers.

After downloading the CICMaldroid2020 and CICMalAnal2017 datasets, 97.3% and 97.6% of the samples were successfully converted to the .wav audio format, respectively. The remaining samples were discarded because they were either corrupted or duplicates. On average, converting CICMaldroid2020 apks to audio took 21 minutes, whereas CICMalAnal2017 took 5 minutes and 34 seconds. Bark-frequency cepstrum coefficient features were generated from the audio files for analysis. The sample data in Figure 3 shows the generated audio files and their corresponding extracted bark frequency features.

We implemented the proposed strategy on a 1.80GHz Intel(R) Core (TM) i7-8565U CPU laptop with 24 GB RAM. The code to implement the machine learning pipeline discussed above was developed using TensorFlow and Python. In the experiment, 23 machine learning classifiers were implemented for performance evaluation. These include extra trees, Gaussian Process, Multi-Layer Perceptron, Bayesian

Network, Passive Aggressive, Support Vector Machine, AdaBoost, Random Forest, KNeighbors, and Decision Tree.

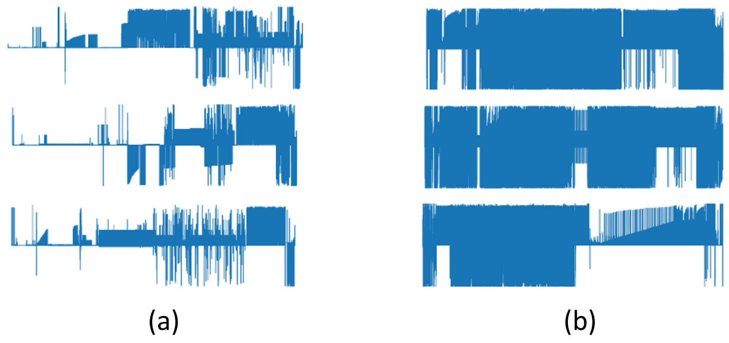

Figure 3. Sample Data (a) Malware waveforms, (b) Benign waveforms

4.1. Classification performance

4.1.1. CICMalAnal2017

To comprehensively measure how accurately bark-frequency cepstral coefficients can be used to classify android applications, classification accuracy, precision, recall, and area under the curve are calculated. Moreover, the train and test times are also calculated to estimate the complexity of the algorithms. The following table shows the results of the malware detection and classification experiments on the CICMalAnal2017 dataset using an 80 – 20 % split.

Table 3. CICMalAnal2017 performance results.

Classifier used	Test Accuracy	Precision	Recall	F1 Score	AUC	Train Time [s]	Test Time [s]
RandomForest	0.9325	0.8434	0.8235	0.8333	0.8921	2.4956	0.0642
CatBoost	0.9301	0.8500	0.8000	0.8242	0.8818	4.0229	0.0034
ExtraTrees	0.9277	0.8873	0.7412	0.8077	0.8585	0.1654	0.0194
LGBM	0.9253	0.8293	0.8000	0.8144	0.8788	0.1860	0.0141
Bagging	0.9229	0.8354	0.7765	0.8	0.8685	0.1464	0.0023
XGB	0.9157	0.7976	0.7882	0.7929	0.8684	0.6426	0.0034
GaussianProcess	0.9157	0.8906	0.6706	0.7651	0.8247	2.6927	0.0171
XGBRF	0.9108	0.8333	0.7059	0.7805	0.8348	0.2434	0.0038
Gradient Boosting	0.9108	0.8000	0.7529	0.7643	0.8522	0.6347	0.0013
DecisionTree	0.9060	0.7738	0.7647	0.7285	0.8536	0.0358	0.0005
KNeighbors	0.9012	0.7558	0.7647	0.7602	0.8505	0.0042	0.0306

Classifier used	Test Accuracy	Precision	Recall	F1 Score	AUC	Train Time [s]	Test Time [s]
SVC	0.8988	0.8525	0.6118	0.7529	0.7922	0.2593	0.0101
MLP	0.8988	0.8413	0.6235	0.7123	0.7966	2.2036	0.0006
AdaBoost	0.8988	0.8209	0.6471	0.7152	0.8053	0.2455	0.0095
LinearSVC	0.8747	0.7619	0.5647	0.7134	0.7596	0.0704	0.0003
Logistic RegressionCV	0.8747	0.7538	0.5765	0.6533	0.7640	1.3264	0.0003
NuSVC	0.8627	0.6429	0.7412	0.6395	0.8176	0.1992	0.0069
RidgeCV	0.8627	0.7593	0.4824	0.6707	0.7215	0.0109	0.0003
SGD	0.8458	0.6567	0.5176	0.6885	0.7240	0.0073	0.0003
BernoulliNB	0.8410	0.6173	0.5882	0.5899	0.7471	0.0028	0.0006
Passive Aggressive	0.8386	0.8000	0.2824	0.6024	0.6321	0.0023	0.0003
Perceptron	0.8241	0.6500	0.3059	0.416	0.6317	0.0024	0.0002
GaussianNB	0.8193	0.5735	0.4588	0.5098	0.6855	0.0020	0.0006

4.1.2. CICMaldroid2020

The following table shows the results of the malware detection and classification experiments on the CICMaldroid2020 dataset using an 80 – 20 % split.

Table 4. CICMaldroid2020 performance results.

MLA used	Test Accuracy	Precision	Recall	F1 Score	AUC	Train Time [s]	Test Time [s]
RandomForest	0.9787	0.9840	0.9885	0.9862	0.9669	18.1183	0.2386
ExtraTrees	0.9784	0.9851	0.9870	0.986	0.9681	0.7067	0.0742
CatBoost	0.9739	0.9809	0.9854	0.9832	0.9601	6.1896	0.0169
KNeighbors	0.9728	0.9820	0.9828	0.9824	0.9607	0.0420	0.1974
XGB	0.9725	0.9805	0.9839	0.9822	0.9587	2.0600	0.0060
LGBM	0.9719	0.9805	0.9831	0.9818	0.9583	0.2125	0.0076
Bagging	0.9686	0.9782	0.9812	0.981	0.9535	1.1242	0.0073
MLP	0.9636	0.9766	0.9762	0.9765	0.9484	17.4428	0.0026
GradientBoosting	0.9609	0.9733	0.9762	0.9748	0.9425	5.3028	0.0062
GaussianProcess	0.9594	0.9758	0.9716	0.9737	0.9448	377.1062	0.8963

XGBRF	0.9571	0.9746	0.9697	0.9722	0.9418	1.5513	0.0065
SVC	0.9553	0.9727	0.9693	0.9721	0.9384	7.2397	0.2970
DecisionTree	0.9526	0.9658	0.9732	0.971	0.9279	0.2303	0.0007
AdaBoost	0.9464	0.9620	0.9690	0.9655	0.9193	1.2990	0.0272
Logistic RegressionCV	0.9464	0.9644	0.9663	0.9654	0.9225	1.4583	0.0004
LinearSVC	0.9449	0.9640	0.9648	0.9644	0.9211	0.5492	0.0008
Perceptron	0.9426	0.9618	0.9640	0.9629	0.9168	0.0119	0.0006
SGD	0.9417	0.9657	0.9586	0.9626	0.9213	0.0631	0.0005
RidgeCV	0.9233	0.9431	0.9586	0.9508	0.8809	0.0227	0.0005
Passive Aggressive	0.9055	0.9712	0.9046	0.9362	0.9067	0.0207	0.0006
BernoulliNB	0.8993	0.9179	0.9552	0.9348	0.8322	0.0059	0.0011
NuSVC	0.8981	0.9247	0.9452	0.9222	0.8416	6.1177	0.1409
GaussianNB	0.6799	0.9458	0.6215	0.7501	0.7501	0.0082	0.0016

The tables 3 and 4 above show the performance statistics of the various models that were evaluated in this study. The tables are sorted by the testing accuracy column. A summary of the relevant top performing algorithms is given below:

- **Accuracy**: Random Forest, extra trees and catboost algorithms achieved the top 3 test accuracy scores in both datasets. The highest scores achieved are 97.9%, 97.8%, and 97.4%, respectively.
- **Precision**: Extra Trees achieved the highest score of 98.5%, followed by random forest which had 98.4%.
- **Recall**: Random forest accomplished the highest recall rate of 98.9 whereas extra trees was second with 98.7%.
- **F1 Score**: Random forest was the best with an f1 score of 98.62%, followed by extra trees which had 98.6%.
- **ROC Score:** Extra trees achieved the best score of 96.8%, followed by random forest, which had 96.7%.
- Generally, it was observed that ensemble algorithms performed well in the malware classification task while the GaussianNB was the worst performing in all experiments. Furthermore, most fast algorithms in terms of processing speed, did not have good results to warranty consideration.

The figure below shows highlights of the top 3 performing algorithms over the two experimental datasets.

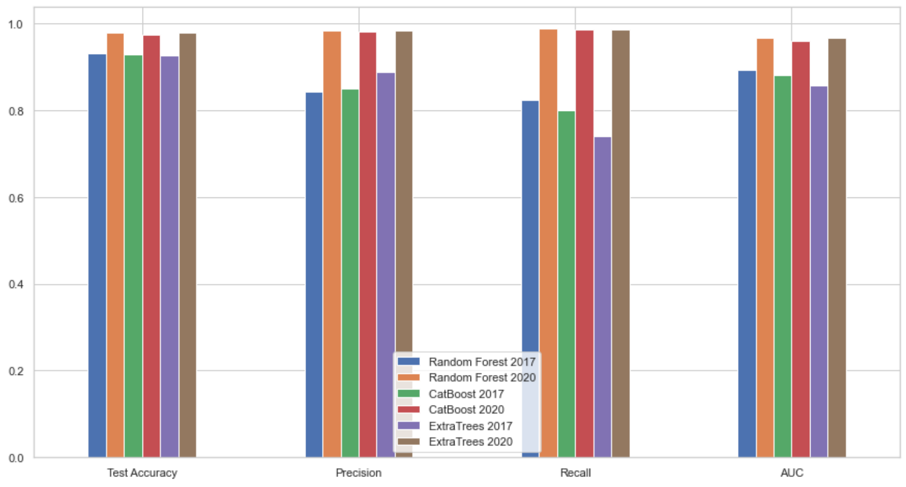

Figure 4. Performance results of the top 3 algorithms

The table 5 below ranks the models column-wise by identifying the best model according to each performance metric.

Table 5. CICMaldroid2020 performance ranking results.

Classifier used	Test Accuracy	Precision	F1 Score	AUC	Train Time [s]	Test Time [s]
RandomForest	**1**	2	**1**	2	22	21
ExtraTrees	2	**1**	2	**1**	11	18
CatBoost	3	4	3	4	19	16
KNeighbors	4	3	4	3	6	20
XGB	5	5	5	5	16	11
LGBM	6	6	6	6	8	15
Bagging	7	7	7	7	12	14
MLP	8	8	8	8	21	10
GradientBoosting	9	11	9	10	17	12
GaussianProcess	10	9	10	9	23	23
XGBRF	11	10	11	11	15	13
SVC	12	12	12	12	20	22
DecisionTree	13	14	13	13	9	6
AdaBoost	14	18	15	17	13	17
LogisticRegressionCV	15	16	14	14	14	1
LinearSVC	16	17	16	16	10	7
Perceptron	17	19	17	18	3	4
SGD	18	15	18	15	7	3
RidgeCV	19	21	19	20	5	2
PassiveAggressive	20	13	20	19	4	5
BernoulliNB	21	23	21	22	1	8
NuSVC	22	22	22	21	18	19
GaussianNB	23	20	23	23	23	9

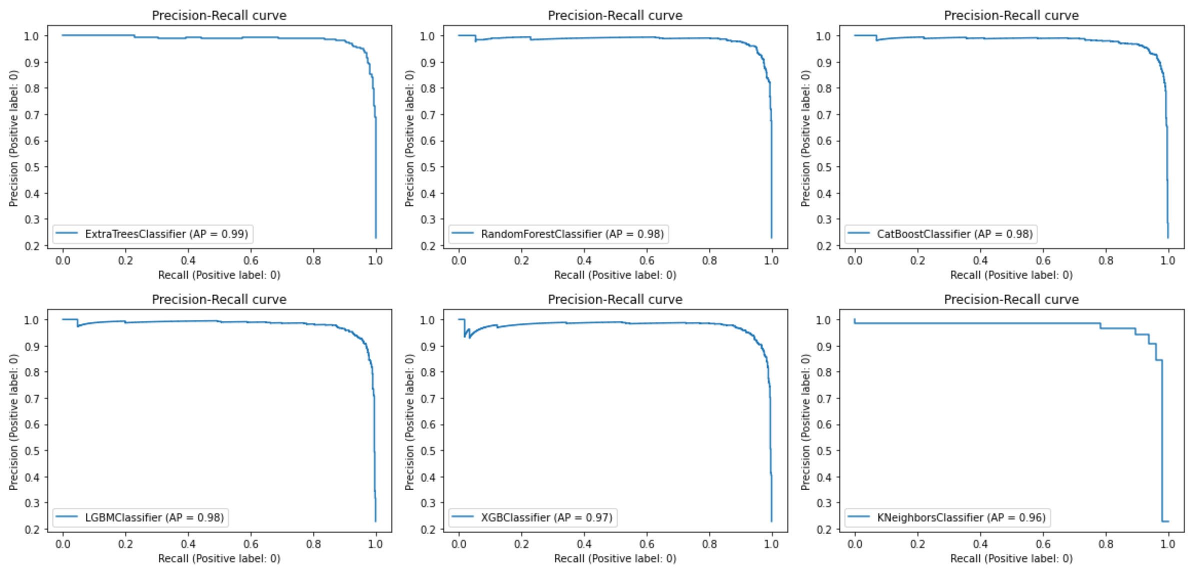

Figure 5. Precision-Recall Graphs of the top algorithms

Based on the highest test accuracy results, the top five models are Random Forest, Extra Trees, CatBoost, KNeighbors, and XGBClassifier. In the precision-recall graphs shown in figure 5, these algorithms have high precision and recall rates, meaning they returned many correctly labelled results. While the differences between the performance metrics of random forest and extra trees are minimal, if one considers the resource limitations of mobile devices, the extra trees algorithm is probably the better choice because of shorter train and test times. Furthermore, it also has a better precision score, meaning it has the highest ratio of malicious applications that are correctly classified. By looking at the two datasets that were used for experimentation, it can also be observed that the highest accuracy achieved improved by around 4.62% by utilising a slightly bigger dataset. It should be noted that the dataset used is substantially smaller than the bigger datasets used in the research field. The achieved results evidently show that bark-frequency cepstral coefficients are promising static features for malware detection.

5. Conclusion

This paper proposes an android malware detection system that uses acoustic signals and Bark Frequency Cepstral Coefficients as malware features. To the best of our knowledge, this is the first study to introduce such features for malware detection. Twenty three machine learning algorithms were used to evaluate the efficiency of the proposed system. As this research has shown, bark frequency cepstral coefficients proved highly discriminative in android malware detection reaching an average precision of 99%.

References

[1] S. O'Dea, "Smartphone subscriptions worldwide 2027 | Statista," 2022. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ (accessed Aug. 02, 2022).

[2] Federica Laricchia, "Global mobile OS market share 2012-2022 | Statista," *Market share of mobile operating systems worldwide 2012-2022*, 2022. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ (accessed Aug. 02, 2022).

[3] Symantec, "ISTR Internet Security Threat Report Volume 24 |," 2019.

[4] M. Yang and Q. Wen, "Detecting android malware by applying classification techniques on images patterns," *2017 2nd IEEE International Conference on Cloud Computing and Big Data Analysis, ICCCBDA 2017*, pp. 344–347, Jun. 2017, doi: 10.1109/ICCCBDA.2017.7951936.

[5] A. Razgallah, R. Khoury, S. Hallé, and K. Khanmohammadi, "A survey of malware detection in Android apps: Recommendations and perspectives for future research," *Computer Science Review*, vol. 39, p. 100358, Feb. 2021, doi: 10.1016/J.COSREV.2020.100358.

[6] D. Ö. Şahın, O. E. Kural, S. Akleylek, and E. Kiliç, "New results on permission based static analysis for Android malware," 2018. Accessed: Aug. 01, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/8355377/

[7] N. Milosevic, A. Dehghantanha, and K. K. R. Choo, "Machine learning aided Android malware classification," *Computers and Electrical Engineering*, vol. 61, pp. 266–274, Sep. 2017, doi: 10.1016/j.compeleceng.2017.02.013.

[8] M. Ganesh, P. Pednekar, P. Prabhuswamy, D. S. Nair, Y. Park, and H. Jeon, "CNN-Based Android Malware Detection," *Proceedings - 2017 International Conference on Software Security and Assurance, ICSSA 2017*, pp. 60–65, 2018, doi: 10.1109/ICSSA.2017.18.

[9] A. Mahindru and A. L. Sangal, "FSDroid:- A feature selection technique to detect malware from Android using Machine Learning Techniques: FSDroid," *Multimedia Tools and Applications*, pp. 1–53, Sep. 2021, doi: 10.1007/s11042-020-10367-w.

[10] O. E. Kural, D. Ö. Şahin, S. Akleylek, and E. Kılıç, "Permission Weighting Approaches in Permission Based Android Malware Detection," 2019. Accessed: Aug. 01, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/8907187/

[11] S. J. K., S. Chakravarty, and R. K. V. P., "Feature Selection and Evaluation of Permission-based Android Malware Detection," 2020. Accessed: Aug. 01, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9142929/

[12] S. Kumar, D. Mishra, and S. K. Shukla, "Android Malware Family Classification: What Works-API Calls, Permissions or API Packages?," *Proceedings - 2021 14th International Conference on Security of Information and Networks, SIN 2021*, 2021, doi: 10.1109/SIN54109.2021.9699322.

[13] E. Amer, "Permission-Based Approach for Android Malware Analysis Through Ensemble-Based Voting Model," 2021. Accessed: Aug. 01, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9447675/

[14] F. Guyton, W. Li, L. Wang, and A. Kumar, "Android Feature Selection based on Permissions, Intents, and API Calls," *2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA)*, pp. 149–154, May 2022, doi: 10.1109/SERA54885.2022.9806471.

[15] S. R. T. Mat, M. F. A. Razak, M. N. M. Kahar, J. M. Arif, and A. Zabidi, "Applying Bayesian probability for Android malware detection using permission features," 2021. Accessed: Aug. 01, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9537090/

[16] M. Upadhayay, A. Sharma, G. Garg, and A. Arora, “RPNDroid: Android Malware Detection using Ranked Permissions and Network Traffic,” 2021. Accessed: Aug. 01, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9513992/

[17] H. H. R. Manzil and M. S. Naik, “COVID-Themed Android Malware Analysis and Detection Framework Based on Permissions,” *2022 International Conference for Advancement in Technology, ICONAT 2022*, 2022, doi: 10.1109/ICONAT53423.2022.9726024.

[18] P. R. K. Varma, K. P. Raj, and K. V. S. Raju, “Android mobile security by detecting and classification of malware based on permissions using machine learning algorithms,” 2017. Accessed: Aug. 01, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/8058358/

[19] T. Lu and S. Hou, “A Two-Layered Malware Detection Model Based on Permission for Android,” 2018. Accessed: Aug. 01, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/8542215/

[20] M. Guri, Y. Poliak, B. Shapira, and Y. Elovici, “JoKER: Trusted detection of kernel rootkits in android devices via JTAG interface,” *Proceedings - 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2015*, vol. 1, pp. 65–73, Dec. 2015, doi: 10.1109/TRUSTCOM.2015.358.

[21] F. Wenbo, Z. Linlin, W. Chenyue, H. Yingjie, Y. Yuaner, and Z. Kai, “AMC-MDL: A Novel Approach of Android Malware Classification using Multimodel Deep Learning,” *Proceedings - IEEE 18th International Conference on Dependable, Autonomic and Secure Computing,* , pp. 251–256, Aug. 2020.

[22] V. Sihag, M. Vardhan, and P. Singh, “BLADE: Robust malware detection against obfuscation in android,” *Forensic Science International: Digital Investigation*, vol. 38, p. 301176, Sep. 2021, doi: 10.1016/j.fsidi.2021.301176.

[23] S. Lee, W. Jung, S. Kim, and E. T. Kim, “Android Malware Similarity Clustering using Method based Opcode Sequence and Jaccard Index,” *ICTC 2019 - 10th International Conference on ICT Convergence: ICT Convergence Leading the Autonomous Future*, pp. 178–183, Oct. 2019, doi: 10.1109/ICTC46691.2019.8939894.

[24] H. Rathore, S. K. Sahay, and S. Thukral, “Detection of Malicious Android Applications: Classical Machine Learning vs. Deep Neural Network Integrated with Clustering ficiency (training and testing time) many folds without much penalty on effectiveness of detection model.” 2021. [Online]. Available: https://developer.android.com/guide/platform

[25] A. Merlo, M. Migliardi, and P. Fontanelli, “On energy-based profiling of malware in Android,” *Proceedings of the 2014 International Conference on High Performance Computing and Simulation, HPCS 2014*, pp. 535–542, Sep. 2014, doi: 10.1109/HPCSIM.2014.6903732.

[26] Y. Dai, H. Li, Y. Qian, and X. Lu, “A malware classification method based on memory dump grayscale image,” *Digital Investigation*, vol. 27, pp. 30–37, Sep. 2018, doi: 10.1016/j.diin.2018.09.006.

[27] M. Damshenas, A. Dehghantanha, K.-K. R. Choo, and R. Mahmud, “M0Droid: An Android Behavioral-Based Malware Detection Model,” *Journal of Information Privacy and Security*, vol. 11, no. 3, pp. 141–157, Sep. 2015, doi: 10.1080/15536548.2015.1073510.

[28] V. M. Afonso, M. F. de Amorim, A. R. A. Grégio, G. B. Junquera, and P. L. de Geus, “Identifying Android malware using dynamically obtained features,” *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 1, pp. 9–17, Sep. 2015, doi: 10.1007/s11416-014-0226-7.

[29] A. Dhavlle and S. Shukla, “A Novel Malware Detection Mechanism based on features extracted from converted Malware binary images,” 2021.

[30] L. Yang and J. Liu, “TuningMalconv: Malware Detection with Not Just Raw Bytes,” *IEEE Access*, vol. 8, pp. 140915–140922, 2020, doi: 10.1109/ACCESS.2020.3014245.

[31] F. Mercaldo and A. Santone, “Audio signal processing for Android malware detection and family identification,” *Journal of Computer Virology and Hacking Techniques,* vol. 17, no. 2, pp. 139–152, 2021, doi: 10.1007/s11416-020-00376-6.

[32] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, “Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification,” *Proceedings - International Carnahan Conference on Security Technology*, vol. 2018-October, Dec. 2018, doi: 10.1109/CCST.2018.8585560.

[33] S. Mahdavifar, A. F. Abdul Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, “Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning,” *Proceedings - IEEE 18th International Conference on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress*, pp. 515–522, Aug. 2020, doi: 10.1109/DASC-PICOM-CBDCOM-CYBERSCITECH49142.2020.00094.

[34] S. Mahdavifar, D. Alhadidi, and A. A. Ghorbani, “Effective and Efficient Hybrid Android Malware Classification Using Pseudo-Label Stacked Auto-Encoder,” *Journal of Network and Systems Management*, vol. 30, no. 1, Jan. 2021, doi: 10.1007/S10922-021-09634-4.

[35] C. Kumar, F. Ur Rehman, S. Kumar, A. Mehmood, and G. Shabir, “Analysis of MFCC and BFCC in a speaker identification system” *2018 International Conference on Computing, Mathematics and Engineering Technologies: Invent, Innovate and Integrate for Socioeconomic Development, iCoMET 2018 - Proceedings*, vol. 2018-January, pp. 1–5, Apr. 2018, doi: 10.1109/ICOMET.2018.8346330.

[36] T. Gulzar, A. Singh, and S. Sharma, “Comparative Analysis of LPCC, MFCC and BFCC for the Recognition of Hindi Words using Artificial Neural Networks,” *International Journal of Computer Applications*, vol. 101, no. 12, pp. 22–27, Sep. 2014, doi: 10.5120/17740-8271.