Online at www.e-iph.co.uk
Indexed in Clarivate Analytics WoS,

E-B
Environment - Behaviour Proceedings Journal

CSSR 2023
https://cssr.uitm.edu.my/2023/

e-IPH
e-International Publishing House Ltd., United Kingdom

10th International Conference on Science & Social Research
Virtual Conference
6 - 7 Nov 2023
Organised by: Research Nexus UiTM (ReNeU), Universiti Teknologi MARA

Android Malware Detection using Deep Learning Classification Approach

Mohd Faris Mohd Fuzi[1*], Nur Amirah Amri[1], Mohammad Hafiz Ismail[1], Tajul Rosli Razak[2]
**Corresponding Author*

[1] Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Perlis Branch, Arau Campus, 02600 Arau, Perlis, Malaysia
[2] Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, 40450 Shah Alam, Selangor, Malaysia

farisfuzi@uitm.edu.my, ab183839@gmail.com, mohammadhafiz@uitm.edu.my, tajulrosli@uitm.edu.my,
Tel: +6 0124169616

Abstract
Android devices are increasingly popular, and the number of threats to Android users is growing. This paper discusses Android malware detection using a deep learning classification approach. This study used malware analysis tools to analyse Android software, extract selected features, and compile the results into a CSV file. Subsequently, we analysed its use in CNN and RNN models for malware detection by measuring standard accuracy. In the development process, CNN outperforms RNN in detecting Android malware, achieving 96 per cent accuracy, whereas RNN achieves 75 per cent.

Keywords: Android Malware, Malware Detection, Deep Learning, Classification Approach

DOI: https://doi.org/10.21834/e-bpj.v10iSI40.7714

1.0 Introduction

People are using smartphones more and more these days. Jay (2023) says that there were about 3.8 billion people who used mobile devices in 2021. In the next few years, this figure is likely to keep going up. According to a survey done by Statcounter Global Stats in 2023, more than 72% of people who use mobile devices utilise Android. Khalil (2025) said that 300,000 new types of malware are made every day. This number indicates how quickly dangers are changing, which is a nightmare. It got a lot of attention from Android users, and about 560,000 new Android malware items were found, such as viruses, Trojans, ransomware, spyware, and adware (Chinnasamy, 2025). Many factors may make the Android system more attractive to cyber attackers. Thus, Android must update and enhance its operating system by providing robust safeguards for its users. However, many users still use the older firmware version and do not update to the newest version. These issues create potential for security threats (Kushwaha et al., 2023).

Android users face additional threats, as malware authors increasingly exploit vulnerabilities in Android devices to enable malicious behaviour. Furthermore, given the proliferation of mobile devices and their associated app stores, the volume of new applications is too large to manually examine each one for malicious behaviour. Once the victim downloads the application from an app store, the attacker may gain access via an infected application (Chia, 2017; Viennot & Nieh, 2014). Thus, malware detection is becoming increasingly complex. Many detection methods have been proposed in the past few decades (Tayyab et al., 2022). Malware detection studies utilising machine learning are increasingly popular because they are a successful strategy for achieving high detection accuracy (Mat, 2021).

eISSN: 2398-4287 © 2025. The Authors. Published for AMER by e-International Publishing House, Ltd., UK. This is an open access article under the CC BY-NC-ND license (http://c5eativecommons.org/licenses/by-nc-nd/4.0/). Peer–review under responsibility of AMER (Association of Malaysian Environment-Behaviour Researchers
DOI: https://doi.org/10.21834/e-bpj.v10iSI40.7714

Some researchers try to reveal the operators and procedures of Android malware. However, most detect it using traditional machine-learning methods. Many ML algorithms have been used, like support vector machines (SVM) (Hyoil et al., 2020; Singh et al., 2022; Droos et al., 2022), K-nearest neighbour (KNN) (Baldini & Geneiatakis, 2019; Assegie, 2021), Bayesian estimation (Castillo-Zúñiga et al., 2020; Yilmaz et al., 2022), genetic algorithms (Yildiz & Dogru, 2019), etc., to build malware detection systems. Researchers use unsupervised learning methods when labelled data are unavailable; in this approach, the ML algorithm learns to distinguish between malware and benign samples. However, some studies have combined supervised and unsupervised learning methods (Arora, 2019).

The most widely used detection technique is Deep Packet Inspection (DPI) (Ghosh & Senthilrajan, 2019), which distinguishes between benign programs and malware by analysing whether each network packet's payload contains sensitive information. This technique can effectively identify abnormal traffic, and many Network Intrusion Detection Systems (NIDS) have adopted it. However, this traditional technique not only requires continuous maintenance of detection rules but also requires payload content detection (Wu, 2017), which often results in a high False Negative Rate (FNR) and threatens user privacy.

To improve the accuracy of malware detection, Shiqi (2019) proposed a neural network-based method. The proposed neural network processes mixed data, including API method-call information and a list of permissions from an Android application. The results indicate an average accuracy of 93% and a false-positive rate of 3.3%.

This paper aims to develop a deep learning classification model using features extracted from Android malware samples and to evaluate its accuracy in Android malware detection. Specifically, the objectives are to analyse the Android malware dataset, design and implement classification models based on Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), and evaluate their accuracy for Android malware detection. Moreover, the paper focuses on analysing the Android Malware family dataset and detecting malware within a deep learning framework. The remainder of the paper is structured as follows: Section 2 reviews related work on deep learning-based malware detection, and Section 3 explains the methodology. Section 4 discusses the results and findings, while Section 5 concludes the paper.

2.0 Literature Review

Iqubal et al. (2024) previously suggested a machine learning technique for identifying Android malware, emphasising app permissions to train models for app classification as benign or dangerous. To check how reliable these models are, they are trained and tested on a large dataset. The research looks at different machine learning algorithms to see which one is better at finding malware. It does this by looking at how accurate each one is. The study emphasises performance, efficiency, and scalability in Android security solutions. A Random Forest got 97.20% accuracy using NATICUSdroid data, which makes it a good choice for Android threat detection.

Meanwhile, Wang et al. (2019) also propose a CNN-based system for detecting Android malware. In this study, the researchers used static analysis tools to extract 1003 static features. They transformed the features of each sample into a two-dimensional matrix as input to the CNN model. The experiment's outcomes demonstrate that the CNN achieves a detection accuracy of 99.68%. However, the researchers studied only some of the static properties in this experiment.

Another study, conducted by Mbaziira et al. (2020), proposed a multi-layer DNN with a classifier (DL4J) to detect and predict the presence of Android malware. The study found that adding layers to the models improved malware detection rates. The results of this study show that DNN not only performs well in mobile malware detection but also outperforms other approaches. Organisations can deploy DNN in their computer networks to protect servers, computers, and cloud-hosted data. The problem with this approach is that the DNN did not eliminate the risk of malware entering the device.

The following study is from Qiu et al. (2021). They had reviewed the literature on advances in deep learning-based Android malware detection. It gives readers a brief, in-depth summary that helps them comprehend the area. Instead of doing an empirical investigation like ours, they wrote a literature evaluation of several deep learning classifiers and future extraction methods. On the other hand, our study wants to compare different types of features and classifiers that researchers utilise on a standard benchmark.

Finally, a study by Nicheporuk et al. (2020) has proposed a convolutional neural network mixed-data model-based approach to Android virus detection. This dataset comprises Android app permissions and API method calls. The approach represents API calls as vectors using Word2Vec, generating feature vectors for semantically comparable API requests. The method encodes each distinct permission as a binary feature indicating its presence or absence in the input sequence, yielding a set of permissions.

3.0 Methodology

The research process consisted of five phases: generating the Android dataset, analysing malware, visualising malware, training and validating the model, and evaluating model performance. Figure 1 shows the flowchart for the Android malware detection model.

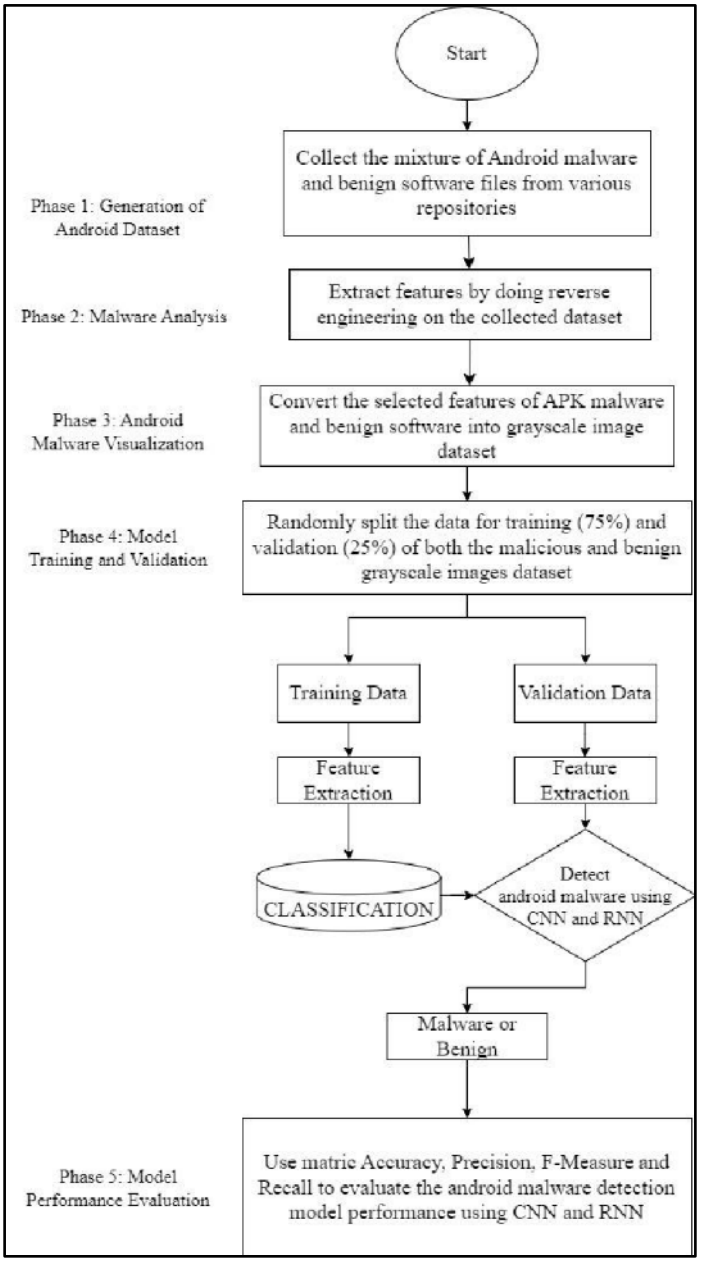


Fig. 1: Android malware detection model flowchart

3.1 Generation of Android Dataset

This phase describes the process of generating Android datasets. The flow in this phase is to set up a secure environment on a physical machine and collect data comprising Android malware and benign software.

3.1.1 Setup Secure Environment

This project used a desktop with an Intel Core i9 CPU and an NVIDIA TURBO GPU as a platform for installing and downloading all necessary software. We installed VirtualBox 7.0, ran Windows 10 on it, and configured Flare VM to create a virtual machine for malware analysis. This safe, protected laboratory environment prevents dangerous malware from breaching the virtualised setup and infecting the underlying host machine. Figure 2 shows the setup for creating a secure environment topology.

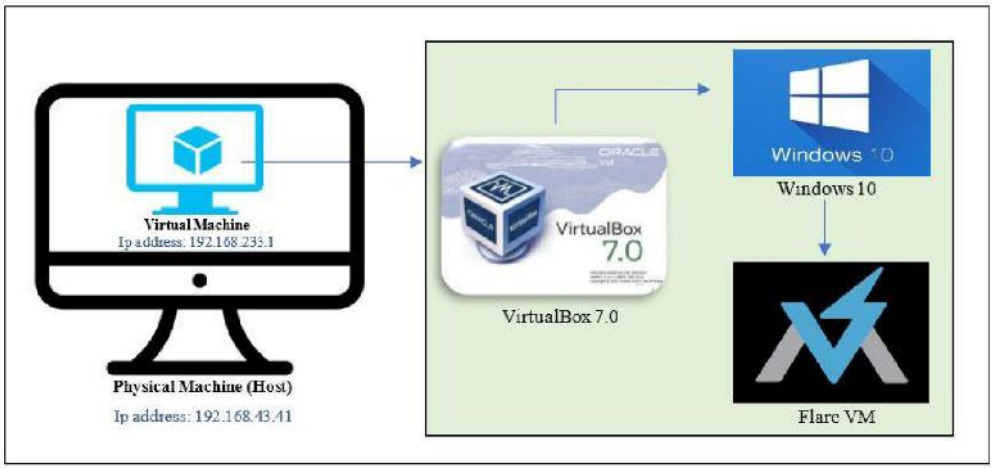


Fig. 2: Secure environment topology

3.1.2 Dataset Collection

Evaluation of deep learning models requires a training dataset. Using a well-known dataset for training is advisable. However, the most significant challenge researchers face is the lack of a reliable dataset. This field currently lacks high-quality datasets. The dataset for this study was sourced from en.softonic.com and rawapk.com. This dataset comprised 500 unique benign and 500 unique malicious APK samples, for a total of 1000 samples. Figures 3 and 4 show samples of benign and malicious software.

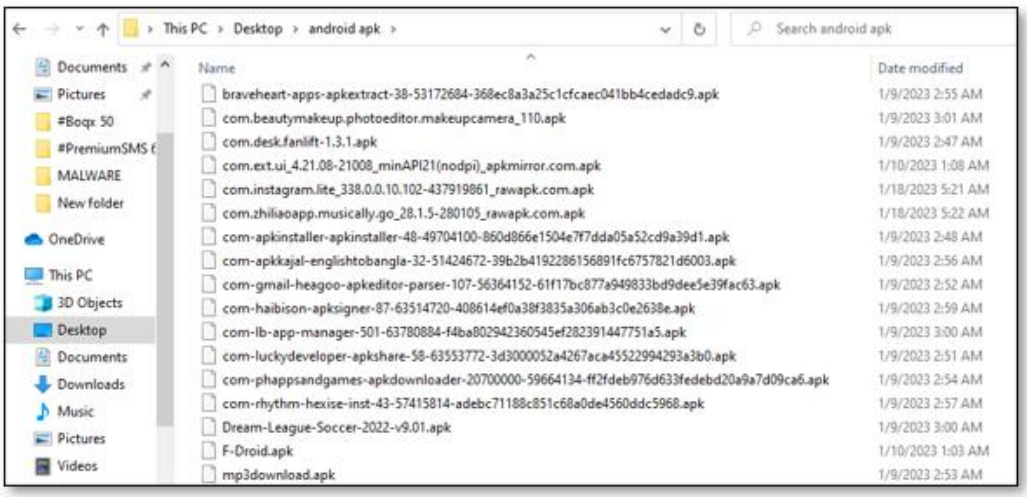


Fig. 3: Samples of benign software

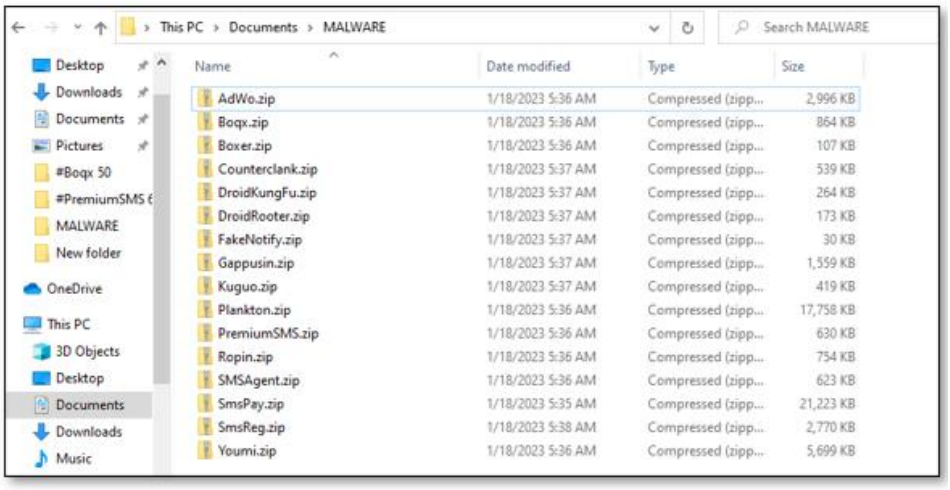


Fig. 4: Samples of malicious software

3.2 Malware Analysis

During this phase, static analysis techniques identified and analysed suspicious files on endpoints and within networks. The procedure included data preprocessing.

3.2.1 Data Preprocessing

The process visualised malware binaries as grayscale images, revealing that images from the same malware family tend to look remarkably similar. This pattern holds for many different malware families, as new malware is often just an updated version of an existing type. Therefore, variants contain approximately the same amount of content and share a very similar visual presentation. The main idea is to transform each malware program into a grayscale image. Based on the previously collected features, the approach constructed an image descriptor to categorise malware using artificial intelligence methods. Figure 5 shows the conversion process of an Android program in APK format into a grayscale image.

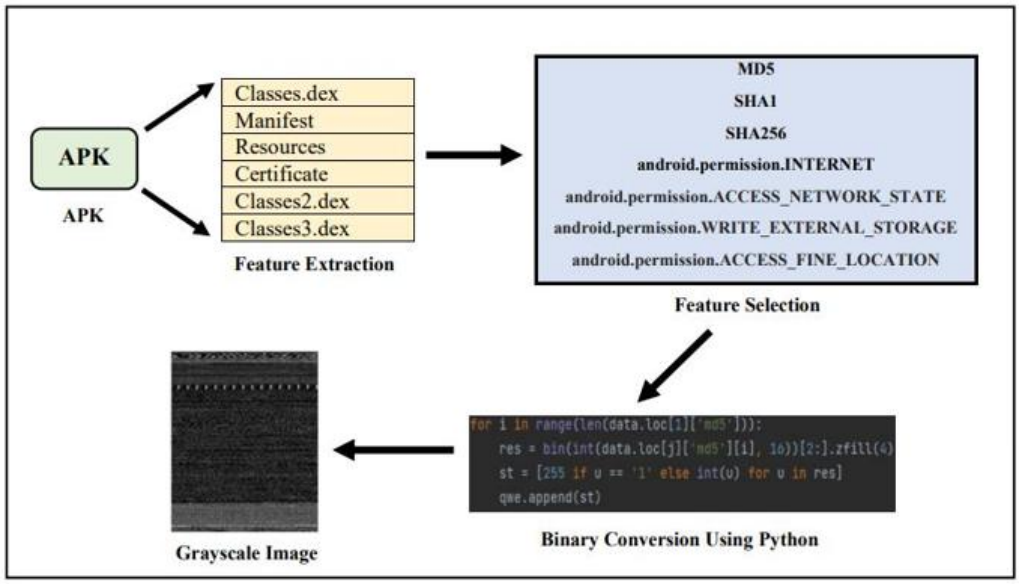


Fig. 5: Conversion process of APK into a grayscale image

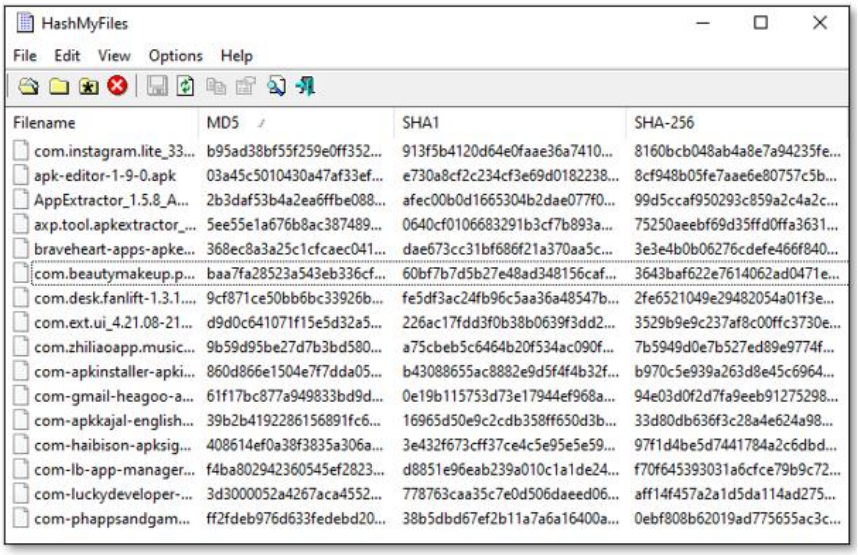
3.2.2 Feature Extraction

An Android application package (APK) is a compressed archive that contains the application's classes: the dex file, the AndroidManifest.xml file, compressed resources, and libraries. The zip file's contents are not human-readable because the Java classes are compressed into a single Dalvik executable (.dex) file. Nevertheless, it is possible to reverse-engineer an APK file to extract the Java source code and associated files using various tools. Figure 6 shows that APKTool was used to extract the .dex and AndroidManifest.xml files from the APK.

```
C:\Mirah\android apk>apktool d com-lb-app-manager-501-63780884-f4ba802942360545ef282391447751a5.apk
I: Using Apktool 2.7.0 on com-lb-app-manager-501-63780884-f4ba802942360545ef282391447751a5.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\fyp12\AppData\Local\apktool\framework\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Baksmaling classes2.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
I: Copying META-INF/services directory
```

Fig. 6: Feature extraction of APK packages using APKTool

The following process involved extracting the APK file's hash. The hash values extracted were MD5, SHA1, and SHA256. Figure 7 shows the hash value extracted from the HashMyFiles tool. These values were collected and exported to a Comma-Separated Values (CSV) file. Android application permissions offer apps access to the phone's hardware and data, as well as the power to operate it. Permissions might be valid or dangerous. For instance, if an application requests permission to access sensitive data, such as the phone book or the camera, it may be malicious or at least suspicious. Therefore, permissions are significant indicators for detecting malicious programs and differentiating them from benign ones.



Filename	MD5	SHA1	SHA-256
com.instagram.lite_33...	b95ad38bf55f259e0ff352...	913f5b4120d64e0faae36a7410...	8160bcb048ab4a8e7a94235fe...
apk-editor-1-9-0.apk	03a45c5010430a47af33ef...	e730a8cf2c234cf3e69d0182238...	8cf948b05fe7aae6e80757c5b...
AppExtractor_1.5.8_A...	2b3daf53b4a2ea6ffbe088...	afec00b0d1665304b2dae077f0...	99d5ccaf950293c859a2c4a2c...
axp.tool.apkextractor_...	5ee55e1a676b8ac387489...	0640cf0106683291b3cf7b893a...	75250aeebf69d35ffd0ffa3631...
braveheart-apps-apke...	368ec8a3a25c1cfcaec041...	dae673cc31bf686f21a370aa5c...	3e3e4b0b06276cdefe466f840...
com.beautymakeup.p...	baa7fa28523a543eb336cf...	60bf7b7d5b27e48ad348156caf...	3643baf622e7614062ad0471e...
com.desk.fanlift-1.3.1....	9cf871ce50bb6bc33926b...	fe5df3ac24fb96c5aa36a48547b...	2fe6521049e29482054a01f3e...
com.ext.ui_4.21.08-21...	d9d0c641071f15e5d32a5...	226ac17fdd3f0b38b0639f3dd2...	3529b9e9c237af8c00ffc3730e...
com.zhiliaoapp.music...	9b59d95be27d7b3bd580...	a75cbeb5c6464b20f534ac090f...	7b5949d0e7b527ed89e9774f...
com-apkinstaller-apki...	860d866e1504e7f7dda05...	b43088655ac8882e9d5f4f4b32f...	b970c5e939a263d8e45c6964...
com-gmail-heagoo-a...	61f17bc877a949833bd9d...	0e19b115753d73e17944ef968a...	94e03d0f2d7fa9eeb91275298...
com-apkkajal-english...	39b2b4192286156891fc6...	16965d50e9c2cdb358ff650d3b...	33d80db636f3c28a4e624a98...
com-haibison-apksig...	408614ef0a38f3835a306a...	3e432f673cff37ce4c5e95e5e59...	97f1d4be5d7441784a2c6dbd...
com-lb-app-manager...	f4ba802942360545ef2823...	d8851e96eab239a010c1a1de24...	f70f645393031a6cfce79b9c72...
com-luckydeveloper-...	3d3000052a4267aca4552...	778763caa35c7e0d506daeed06...	aff14f457a2a1d5da114ad275...
com-phappsandgam...	ff2fdeb976d633fedebd20...	38b5dbd67ef2b11a7a6a16400a...	0ebf808b62019ad775655ac3c...

Fig. 7: Hash extraction from HashMyFiles

Application permissions were extracted from the AndroidManifest.xml file and saved in a CSV file. Figure 8 displays the features selected in the .csv file. The extracted permissions are as follows:

- android.permission.INTERNET
- android.permission.ACCESS_NETWORK_STATE
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.ACCESS_FINE_LOCATION

	md5	sha1	sha256	INTERNET	WRITE_EXTERNAL_STORAGE	ACCESS_NETWORK_STATE	ACCESS_FINE_LOCATION
2	ac6255e5fb6f0a	985ed3cca1fb1728	0ba2abf055b6c109	1	1	0	0
3	befc178a2550d5	4d130440ad0711dc	0bae2e1986bf9895	1	1	1	0
4	4ffaa00f77d75c5	badfbea865ccf7387	0d0b5556d8db065	1	0	0	1
5	5e851b5216812	6ee79aa64d44fdd6	0e83de6d3851bb8	1	1	0	1
6	8d52070201f2a8	21cee13e8c217c4a5	1a4c4178e3646505	1	1	0	1
7	04ee54e6dbf07	38d0c7467eb6b696	1a6e6bca561c4bf6	1	1	0	1
8	f1aa24c1641471	6f336337888fdd57c	1a7128ce758fbdf8	1	1	0	1
9	bf85ce3551ba15	a412fa05e98d53b5	1a9415ffe358a975	1	0	0	1
10	7d2e365f8859b	cb12f61952ecb293	1af18aae81699fb5	1	1	0	1
11	3a1fbde403d4b	4e8bb4e180e6d54f	1c9335ec868f163a1	1	0	0	0
12	3a1fbde403d4b	4e8bb4e180e6d54f	1cbd152993693b12	1	0	0	0
13	7ebdcfe63156d	f7cd210a3629ad925	1f05407d36149e79	1	0	0	0
14	6ae45b47eb736	43cfb8bbbb656c5b	2a95ef7ec999010f2	1	0	0	1
15	3758bb534871e	295d3163021d2349	2a43290321f8ffd3b	1	1	0	0
16	b5fe89c569e1e	f8ce72ac084cc5f75	2b4611729add2872	1	0	1	1
17	bd16a728259f7	3232a901ac75fc7ce	2bad7d74f05967f7	1	0	0	1
18	568a0e80566ce	6fbd82a4444e9f74	2c390aaab77f238d	1	1	0	0
19	fc774d73ccac3b	eeffa476d271a114	2dd727e2b3cd4ad	1	1	1	1
20	6fa6ddf071ac16	2c834a9df48768e1	2f9485bcbb1d4acd	1	0	0	1
21	2ec8a5755e34b	df48841e50aaa98c	3ada67377118a04f	1	0	0	1
22	a4f0c77500010c	0186c2ad6f8c2b9f8	3bd5ef5bc8529f1a	1	1	0	1
23	25dee7418a17b	9d52e200a5e1e28f	3c79cf1fb1d541666	1	0	0	1
24	4330def9470f6e	1bae0d87ce850152	3d054cda9193df9d	1	1	0	1

Fig. 8: Selected features in .csv file

3.3 Malware Visualisation

The process represents the extracted features from the APK dataset as hexadecimal values. Thereafter, the process converted them to binary and then to grayscale. PyCharm was used in this study to convert the binary image to grayscale. The conversion ran in a for loop, and the process will be the same for all features. The Python code generated a grayscale image dataset containing both benign and malicious software. Figure 9 shows the sample dataset of grayscale images. The deep learning process will use the 1000 grayscale images in this dataset. The image will depict the binary representation of each bit derived from the original hexadecimal value. Due to variations in the features, each image has a distinctive quality.

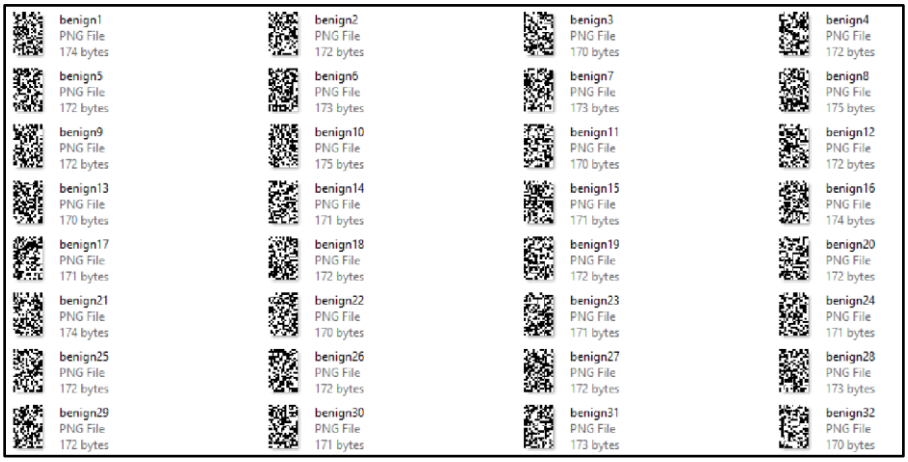


Fig. 9: Benign grayscale image dataset

3.4 Model Training and Validation

This study constructed deep learning classification models using Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) architectures to train and validate Android feature extraction. Both the malicious and benign grayscale image datasets were randomly split into training (75%) and validation (25%) sets. We then evaluated the performance of the resulting Android malware-detection model.

3.4.1 CNN Scripting

Figure 10 displays the training script used to train the APK's benign and malware grayscale images. It will train for the specified number of epochs to learn both image sets.

```
history=model.fit(
    x=training_aug,
    epochs=EPOCHS,
    validation_data=(testX,testY),
    batch_size=training_aug.batch_size,
    steps_per_epoch = training_aug.n // training_aug.batch_size,
    callbacks = cb
)



model.save_weights(MODEL_SAVE_WEIGHT)
model.save(MODEL_SAVE)
```
Fig. 10: CNN training script

Figure 11 shows the validating script used in this study. The validating script displayed the accuracy, precision, recall, F1 scores, and confusion matrix, and, lastly, a classification report detailing the training data.

```
# Adding the first LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 10,activation='tanh', return_sequences = True, input_shape = (X_train.shape[1], 1)))

# Adding a second RNN layer and some Dropout regularisation
regressor.add(SimpleRNN(units = 50,activation='tanh', return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a third RNN layer and some Dropout regularisation
regressor.add(SimpleRNN(units = 50,activation='tanh', return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a fourth RNN layer and some Dropout regularisation
regressor.add(SimpleRNN(units = 50,activation='tanh', return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a fifth RNN layer and some Dropout regularisation
regressor.add(SimpleRNN(units = 50))
regressor.add(Dropout(0.2))
```
Fig. 11: CNN script validation

3.4.2 RNN Scripting

The architecture of the RNN model, comprising many layers, is shown in Figure 12. In various situations, LSTM nodes include a droppable input connection. When input dropout is enabled, each LSTM block does not use input links in node activations or weight updates. As a result, the system saves only information that meets a specified probability threshold.

```
# Compiling the RNN
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error' ,metrics=['accuracy'])


# Fitting the RNN to the Training set
t = regressor.fit(X_train, y_train, epochs = 8, batch_size = 32, validation_data=(X_test,y_test))

print("--- %s seconds ---" % (time.time() - start_time))

X_train.shape
```

Fig. 12: RNN training script

The constructed RNN employed the Adam optimiser, as shown in Figure 13, because it adjusted the learning rate for each network weight. The number of epochs was 8, while the batch size was 32.

```
# Compiling the RNN
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error' ,metrics=['accuracy'])


# Fitting the RNN to the Training set
t = regressor.fit(X_train, y_train, epochs = 8, batch_size = 32, validation_data=(X_test,y_test))

print("--- %s seconds ---" % (time.time() - start_time))

X_train.shape
```

Fig. 13: RNN validating script

3.5 Model Performance Evaluation

The purpose of model evaluation was to determine whether a model's performance corresponds to its objectives. As a result, the appropriate metrics fully assess the efficacy of the Android malware detection or classification model. Existing studies used the following evaluation metrics: accuracy, precision, recall, and F1 score. False Positives (FPs) are essential for an effective Android malware detector, given the above parameters. A high FP indicates the need for additional security professionals to filter out misclassified malicious software. The fundamental objective of an Android malware detector is to identify as many malicious apps as possible. A high false positive rate (FP) indicates ineffective malware detection, as it tends to miss most malware samples. Furthermore, if we fail to detect malware samples and distribute them as benign apps in Android app markets, they could potentially cause severe harm to end users. Often, a strong malware detector has a high F1 score.

4.0 Results and Discussion

This section analyses the results and discussion.

4.1 Result and Analysis for CNN Architecture

Figure 14 presents the confusion matrix results. The graph shows that the system correctly identified 99% of benign datasets as benign software. In this context, the term "true positive rate" applies. The number of benign datasets mistakenly detected as malicious software accounts for 1.2% of the false-positive rate. One could classify this type of event as a false alarm. False positive rate is the percentage of malware samples incorrectly recognised as benign software. It currently sits at 7.7%. The system correctly identifies 92% of malware samples as malicious. "True negative rate" is the phrase for this situation.

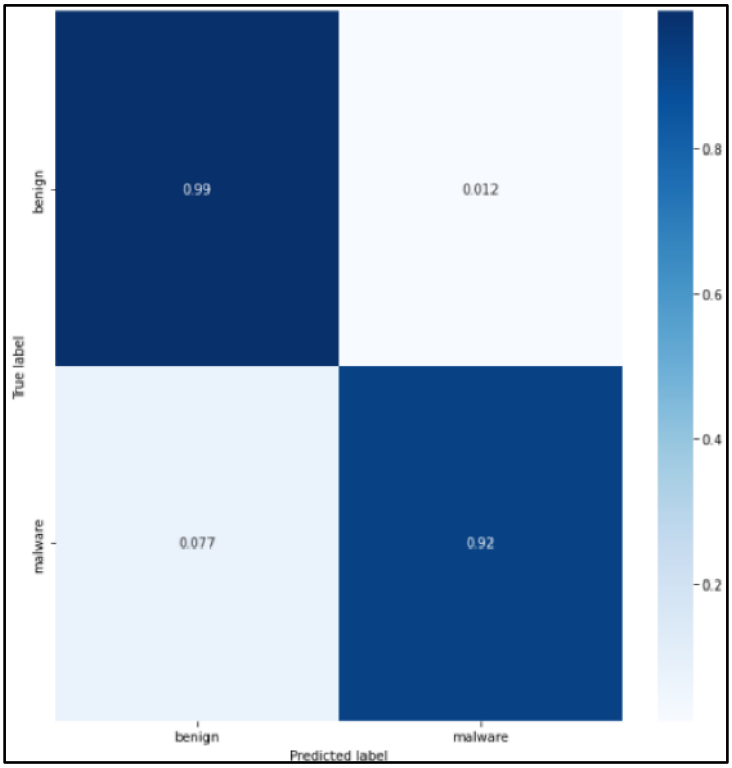


Fig. 14: Result of CNN confusion matrix

Figure 15 displays the accuracy of each validated grayscale image. The green label above the image indicates that the dataset validation was successful, while the red colour indicates that the dataset was incorrectly categorised.

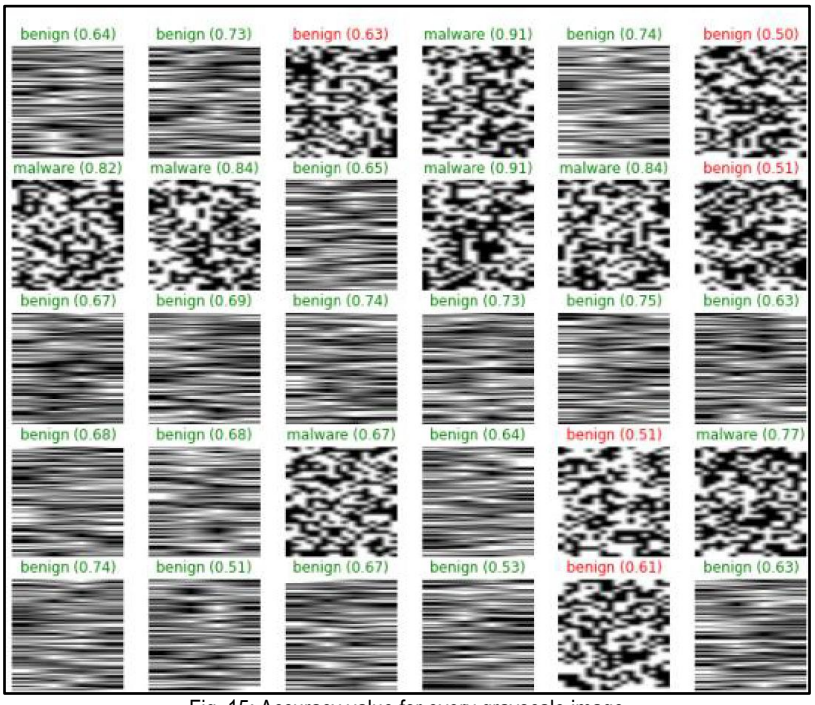


Fig. 15: Accuracy value for every grayscale image

4.2 Result and Analysis for RNN Architecture

Figure 16 displays the confusion matrix graph. This validation set has an actual positive rate of greater than 35%. The negative rate is 40%, showing that the system correctly identified the validation set as benign rather than classifying it as malware. The false-positive rate is less than 10%, which is favourable because the validation set is less likely to be misidentified or misclassified as malware, whereas the feature sample is benign. The false-negative rate is 16.5%, indicating that the validation set misclassifies malware features as benign.

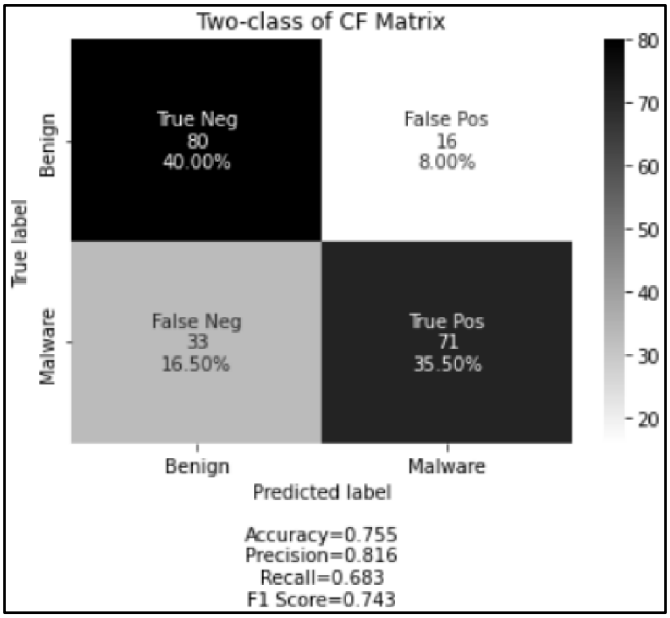


Fig. 16: Result of RNN confusion matrix in a graph

4.3 Comparison between CNN and RNN Models

Table 1 shows that the CNN achieves 96.1% accuracy, whereas the RNN achieves only 75.5%. CNNs are highly effective at classifying both malicious and benign inputs. Limited memory bandwidth constrains RNNs, leading to less efficient use of neural network resources and lower accuracy than CNNs.

Table 1. Comparison between CNN and RNN validation results

Model (%)	CNN	RNN
Accuracy	96	75.5
Precision	96.1	81.6
Recall	96	68.3
F1-Score	95.9	74.3

5.0 Conclusions

The primary accomplishment of this project is the design, development, and evaluation of a deep learning system for detecting Android malware. The CNN and RNN models are used to train the Android feature. The dataset is created by extracting features from Android software, converting them to binary, and transforming them into a grayscale image in Python for use with a CNN. At the same time, the RNN only needs the features extracted in a CSV file. The features were trained in Spyder using a CNN model. This research achieved accuracy through training and validation. 75% of the combined grayscale images of benign and malicious software are allocated to training. 15% of the grayscale image collection is allocated for validation. The outcome is being observed and evaluated with respect to performance and accuracy for both models. The CNN model achieves 96% classification accuracy, whereas the RNN model achieves 75%. Both results are compared and evaluated. The results show that CNNs are more accurate than RNNs at detecting Android malware.

Acknowledgements

The author hereby acknowledges the financial support from University Technology MARA (UiTM) under 600-RMC 5/3/GPM (058/2022).

Paper Contribution to the Related Field of Study

The contribution of this study is to help malware researchers develop systems for detecting Android malware using a deep learning classification approach and to develop effective strategies to enhance system security. This study is also beneficial to future researchers as a reference or guideline for detecting well-known threats, such as Android malware, using a deep learning approach, as they need to keep up with the latest developments and trends in cybersecurity.

References

Arora, A., Peddoju, S.K., Chouhan, V., & Chaudhary, A. (2019). Hybrid Android malware detection by combining supervised and unsupervised learning. 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18). Association for Computing Machinery. pp. 798–800. https://doi.org/10.1145/3241539.3267768.

Assegie, T.A. (2021). An Optimised KNN Model for Signature-Based Malware Detection. International Journal of Computer Engineering in Research Trends (IJCERT). Vol. 8, Issue 02, pp. 46–49, https://ijcert.org/ems/ijcert_papers/V8I206.pdf.

Baldini, G., & Geneiatakis, D. (2019). A Performance Evaluation on Distance Measures in KNN for Mobile Malware Detection. 6th International Conference on Control, Decision, and Information Technologies (CoDIT). pp. 193–198. https://doi.org/10.1109/CoDIT.2019.8820510.

Castillo-Zúñiga, I.; Luna-Rosas, F.; Rodríguez-Martínez, L.; Muñoz-Arteaga, J.; López-Veyna, J.; Rodríguez-Díaz, M.A. (2020). Internet Data Analysis Methodology for Cyberterrorism Vocabulary Detection, Combining Techniques of Big Data Analytics, NLP, and Semantic Web. International Journal on Semantic Web and Information Systems (IJSWIS). 16(1). 69–86. DOI: 10.4018/IJSWIS.2020010104.

Chia, C., Choo, K.-K. & Fehrenbacher, D. (2017). How Cyber-Savvy are Older Mobile Device Users? Mobile Security and Privacy. Advances, Challenges, and Future Research Directions. Pages 67–83. https://doi.org/10.1016/B978-0-12-804629-6.00004-3.

Chinnasamy, V. (2025). 192 cybersecurity statistics for 2025. Indusface Blog. Retrieved August 20, 2025, from https://www.indusface.com/blog/key-cybersecurity-statistics/

Droos, A., Al-Mahadeen, A., Al-Harasis, T., Al-Attar, R. & Ababneh, M. (2022). Android Malware Detection Using Machine Learning. 13th International Conference on Information and Communication Systems (ICICS). pp. 36–41.

Ghosh, A., & Senthilrajan, A. (2019). Research on Packet Inspection Techniques. International Journal of Scientific & Technology Research. 8. 2068-2073.

Hyoil, H., Lim, S., Suh, K., Park, S., Cho, S., & Park, M. (2020). Enhanced Android Malware Detection: An SVM-Based Machine Learning Approach. IEEE International Conference on Big Data and Smart Computing (BigComp). 75–81. http://dx.doi.org/10.1109/BigComp48618.2020.00-96.

Iqubal, A., & Payal, A. (2024). Malware detection technique for Android devices using machine learning algorithms. 2024 International Conference on Computing, Sciences and Communications (ICCSC) (pp. 1–6). IEEE. https://doi.org/10.1109/ICCSC62048.2024.10830310

Jay, A. (2023). Number of Smartphone and Mobile Phone Users Worldwide in 2020/2021: Demographics, Statistics, Predictions. FinancesOnline. https://financesonline.com/number-of-smartphone-users-worldwide. Accessed: 2023-Jan-11.

Khalil, M. (2025). How Many Cyber Attacks Happen per Day in 2025? Accessed: August 20, 2025. [Online]. Available: https://deepstrike.io/blog/how-many-cyberattacks-happen-every-day.

Kushwaha, P. K., Kumar, V., Saraswat, M., Singh, P., Kumar, R., & Vashisht, S. (2023). Android malware detection and its security. 2023 6th International Conference on Contemporary Computing and Informatics (IC3I) (pp. 700–704). IEEE. https://doi.org/10.1109/IC3I59117.2023.10397640

Mat, S.R.T., Ab Razak, M.F., Kahar, M.N.M., Arif, J.M., Mohamad, S. & Firdaus, A. (2021). Towards a Systematic Description of the Field Using Bibliometric Analysis: Malware Evolution. Scientometrics, 126, 2013–2055 https://doi.org/10.1007/s11192-020-03834-6.

Mbaziira, A.V., Diaz-Gonzales, J. & Liu, M. (2020). Deep Learning in Detection of Mobile Malware. Journal of Computing Sciences in Colleges, vol. 36, no. 3, pp. 80–88.

Nicheporuk, A., Savenko, O., Nicheporuk, A., & Nicheporuk, Y. (2020). An Android Malware Detection Method Based on CNN Mixed-data Model. EasyChair Preprint no. 4345. https://easychair.org/publications/preprint/cpxr.

Qiu, J., Zhang, J., Luo, W., Pan, L., Nepal, S., & Xiang, Y. (2021). A Survey of Android Malware Detection with Deep Neural Models. ACM Computing Surveys, 53(6), 1–36. https://doi.org/10.1145/3417978.

Shiqi, L. (2019). Android Malware Analysis and Detection Based on Attention-CNN-LSTM. Journal of Computers, 31–43. https://doi.org/10.17706/jcp.14.1.31-43.

Singh, P., Borgohain, S. K. & Kumar, J. (2022). Performance Enhancement of SVM-based ML Malware Detection Model Using Data Preprocessing. 2nd International Conference on Emerging Frontiers in Electrical and Electronic Technologies (ICEFEET). pp. 1–4, doi: 10.1109/ICEFEET51821.2022.9848192.

Statcounter Global Stats. (2023). Mobile Operating System Market Share Worldwide 2020. Statcounter Global Stats. https://gs.statcounter.com/os-market-share/mobile/worldwide. Accessed: 2023-Jan-11.

Tayyab, U., Khan, F. B., Durad, M. H., Khan, A., & Lee, Y. S. (2022). A Survey of the Recent Trends in Deep Learning Based Malware Detection. Journal of Cybersecurity and Privacy. 2. 800–829. 10.3390/jcp2040041.

Viennot, N., Garcia, E. & Nieh, J. (2014). A Measurement Study of Google Play. 2014 ACM International Conference on Measurement and Modelling of Computer Systems (SIGMETRICS '14). 221–233. https://doi.org/10.1145/2591971.2592003.

Wang, Z., Li, G., Chi, Y., Zhang, J., Yang, T., & Liu, Q. (2019). Android Malware Detection Based on Convolutional Neural Networks. 3rd International Conference on Computer Science and Application Engineering - CSAE 2019. https://doi.org/10.1145/3331453.3361306.

Wu, S. (2017). Detecting Remote Access Trojans through External Control at Area Network Borders. ACM/IEEE Symposium on Architectures for Networking and Communications. 10.1109/ANCS.2017.27.

Yildiz, O. & Dogru, I.A. (2019). Permission-based Android Malware Detection System using Feature Selection with Genetic Algorithm. International Journal of Software Engineering and Knowledge Engineering, Vol. 29, No. 02, pp. 245-262. https://doi.org/10.1142/S0218194019500116.

Yilmaz, A. B., Taspinar, Y. S., & Koklu, M. (2022). Classification of Malicious Android Applications Using Naive Bayes and Support Vector Machine Algorithms. International Journal of Intelligent Systems and Applications in Engineering, 10(2), 269–274. Retrieved from https://www.ijisae.org/index.php/IJISAE/article/view/2010.