Hindawi
Security and Communication Networks
Volume 2022, Article ID 2957203, 9 pages
https://doi.org/10.1155/2022/2957203

Research Article

NLP Technique for Malware Detection Using 1D CNN Fusion Model

Paul Ntim Yeboah[1] **and Haruna Balle Baz Musah**[2]

[1]*Department of Academics, Ghana-India Kofi Annan Centre of Excellence in ICT, Accra, Ghana*
[2]*Department of Research and Innovation, Ghana-India Kofi Annan Centre of Excellence in ICT, Accra, Ghana*

Correspondence should be addressed to Paul Ntim Yeboah; paulny@aiti-kace.com.gh

Received 25 January 2022; Revised 20 May 2022; Accepted 23 May 2022; Published 10 June 2022

Academic Editor: Mohammad Ayoub Khan

With the record of the highest market share of mobile operating systems, the Android operating system has become a prime target for cyber perpetrators as malicious applications are leveraged as attack vectors to exploit Android systems. Machine learning detection solutions that have become a resort mostly rely on handcrafted features, a process deemed to be laborious and time-consuming. In this article, we employ a deep learning-based model consisting of 1-dimensional convolutional neural network (1D CNN) to automate the detection of Android malware. Our choice of 1D CNN was motivated by the computational advantage of 1D convolution operations over 2D CNN. The proposed model automatically extracts features from semantically embedded n-grams of raw static operation code (opcodes) sequences to determine the maliciousness of a binary file. Predictions of the 1D CNN model trained on multiple feature sets of n-gram opcode sequences are combined using a weighted average ensemble. Optimal prediction weights were obtained using a grid search on values in the range 0 to 1. With an Android dataset comprising 4951 malware and 2477 benign samples, our model yielded a positive predictive value of 98% and sensitivity of 97% using a weight parity of 0.5 for ensemble unigram and bigram opcode sequences.

1. Introduction

According to a 2021 report by Statista [1], the Android OS leads the market share of mobile OS by 85.9%; however, this breakthrough in Android operating system patronage comes under the expense of security, as the OS has become keen target for cyber criminals. Data from Lookout Energy [2] on 2020 to 2021 security report shows a 161% spike in mobile phishing attacks. Whiles these phishing campaigns may have different malicious intents like credential theft, most are geared at spreading malware. Unfortunately, existing solutions employed to detect malicious Android applications mostly rely on signature-based methods which usually fall short in generalizing to never-before-seen (zero-day) malware. Research however has shown that such zero-day applications are predominantly variants of already existing malware, generated via code obfuscation techniques like metamorphism. Over the years, metamorphic methods including dead code insertion with no operation instructions, instruction replacement, code transposition, and registry substitution [3, 4] are seen to be leveraged to create functionality-preserving malware variants to circumvent traditional detection engines.

The complexities of tactics leveraged by malware creators have rendered conventional detection solutions almost futile, hence demanding a more sophisticated alternative solution. This demand has motivated researchers to explore data-driven techniques with machine learning, which already have produced state-of-the-art results in malware detection. Machine learning-based methods have proven to be more resilient against malware obfuscation, that is, are able to generalize to never-before-seen semantic-preserving malware variants as opposed to traditional detection solutions. In spite of this breakthrough, features employed to train machine learning models mostly require domain expertise to extract. Such a manual approach to feature extraction can be time-consuming and laborious.

In this research, we propose an automated approach to Android malware detection. We implement end-to-end learning using a deep neural network for malware detection without the need for manual feature engineering. The model takes as input raw data from binary samples and extracts features that are further used to learn malicious and benign patterns. Here, we feed raw unigram and bigram opcode sequences to a deep learning model consisting of a 1-dimensional convolutional neural network to automatically learn representations from the raw input for malware detection. We summarize the main contributions as follows:

(1) To propose end-to-end 1D convolutional neural network-based solution to automate the detection of Android malware.

(2) To investigate the detection efficiency of the proposed model on ensemble n-gram opcode sequences.

2. Related Works

Successes chalked on real-world tasks like computer vision and natural language processing have made AI-powered applications to be ubiquitous. The area of malware detection is no exception as machine learning-based detection solutions have produced state-of-the-art results and proved to be robust against malware obfuscation [5]. To be trained by ML models, Android executables are represented by discriminative features which are mostly obtained either from static or dynamic analysis.

With the static analysis approach, the executable is examined by reverse engineering techniques without executing it. Here, static features including permissions, operation codes, certificates, API function calls, call graphs, raw byte sequences, and special strings [5, 6] are utilized to discriminate malware from benign samples. A malware detection framework proposed by Christiana et al. [7] extracted static features consisting of Android permissions and trained ensemble models with classical machine learning algorithms which obtained an accuracy of 98.16%. Another work, DREBIN, proposed by Arp et al. [8] has also utilized broad static features including permissions, hardware access features, API calls, and network address information to train linear support vector machine (SVM) which yielded a malware detection rate of 94%. Similarly, DroidAPIMiner [9], a lightweight malware detection solution leveraged on API level features to discriminate Android malware from benign at an accuracy of 99% using k-nearest neighbour classifier. Open-source reverse engineering tools including Androguard [10], APKTool [11], and smali [12] are common applications used by researchers for extracting static features.

Dynamic analysis on the other hand requires an execution of the binary file in order to monitor run-time interactions such as API calls, network traffic, memory and register's usage, and instruction traces. A malware classification model, DroidScribe [13], that used run-time features comprising system calls, file system, and network access features was able to classify malware into families with accuracy between 84% and 94%. Dimjašević et al. [14] have shown the efficacy of run-time system call features on their malware classification model after reaching a detection accuracy of 96% on Android samples. DroidRanger, a framework proposed by Zhou et al. [15], also demonstrated how run-time-based permission features can help detect malicious applications from official Android markets.

TaintDroid [16] and DroidSope [17] are common research sandbox tools available for dynamic analysis on Android applications. The benefit of the run-time approach to feature extraction is that it has more resilience against obfuscators like packers; however, complexities associated with dynamic analysis limit deployment to cloud servers since smart mobile phones may not have the capacity for on-device analysis.

The discussed static and dynamic features obtained from executables are required to be further converted into numeric representations before getting used as input to downstream models. Common practices have leveraged on embedding techniques including bag-of-words (BoW) and term frequency-inverse document frequency (TF-IDF) [18]; however, both embedding models fail to capture semantic relationships in texts which inhibits their use in real-world problems. Recently, context-aware embedding techniques like word2vec [19] and GloVe [20] have been combined with deep learning models to automate feature extraction.

A proposed automated malware detection framework MalDozer by Karbab et al. [21] for example represented Android API method sequences in semantic vector forms using word2vec embedding and further employed a 2-dimensional convolutional neural network to extract features to train a model that was able to detect and classify malware at an F1-score of 96%-99%. Similarly, Sun et al. [22] have shown in their proposed malware categorization framework how semantic vectors from word2vec can be combined with temporal convolutional network (TCN) for Windows malware detection. Their proposed word2vec-based TCN framework outperformed one-hot encoding-based TCN.

In this research, we propose a framework similar to the work in [22, 23], where we leverage on 1-dimensional convolutional neural network to train context-aware represented malware features of opcode sequences. The notion is that, unlike other research works like [21, 24, 25] which propose the use of 2D convolutional neural network for malware detection, our work considers the high computational cost associated with 2D convolutions and employs 1D which has low computational complexities to achieve the same task.

3. Background and Motivation

Classical machine learning-based malware detection frameworks predominantly relies on input features manually engineered from models including TF-IDF, Markov chains, principal component analysis (PCA), and information gain (IG). The daunting nature of these models however is that the release of new dataset versions will require manual update of the feature extraction model to refine the input features to the classifier. This manual operation of handcrafted feature engineering is perceived to be time-consuming.

Quite recently, deep learning models have been explored and proven to be a better alternative to the manual approach of feature extraction, especially when dealing with large dataset. Convolutional neural network (CNN) that is arguably one of the most exploited deep learning models are widely used in computer vision and natural language processing task to automate feature extraction for downstream models. State-of-the-art results have been achieved with the CNN approach of automating feature extraction. This boost in performance with CNN networks however comes at the expense of computational complexities. Motivated by the automated feature extraction capability of CNN, we propose a malware detection framework which leverages on CNN to automate representation (feature) learning of raw n-gram opcode sequences. Here, no domain knowledge is required as the CNN model is able to automatically learn and extract low and high-level features from the raw Android binary. In our proposal, we factor in measures to curb potential computational burdens associated with CNNs. First, to keep the CNN model simple, we chose few layers. Second, we adapt to a 1-dimensional CNN, which is reported to be computationally efficient. Adding to that, we propose an ensemble technique which according to recent studies can be leveraged as a defense against adversarial attacks [25]. Strauss et al. in their work [25] have shown how ensemble learning could improve accuracy on CNN task while intrinsically improving resilience against adversarial obfuscation. By this motivation, we propose an ensemble model and focus on evaluating the proposed ensemble model on performance against malware detection, and in future studies, we shall test the model on robustness against adversarial attacks.

4. Proposed Methodology

In this section, we outline the proposed end-to-end model which facilitates the automation of Android malware detection. The model treats Android malware detection as a time-series classification task. Beginning with assembly instruction sequences obtained from Android executables, we leverage on natural language processing (NLP) and deep convolutional neural network to automatically extract representations (features) from the raw unstructured opcodes to learn malicious and benign patterns. We train the model on unigram and bigram opcode feature sets and aggregate predictions from the multi-feature sets via a weighted average ensemble as shown in Figure 1.

4.1. Extraction of Operation Codes Sequence. Android applications are packaged as archived file known as APK which contains program components including: files (classes.dex, AndroidManifest.xml, resources.arsc) and directories (assets, lib, res, META-INF). The APK is analogous to EXE and DEB executables in the Windows and Linux domain, respectively. In this study, we focus on classes.dex, which is the actual executable file of the APK containing compiled Java classes. To obtain assembly operation codes as input to our proposed model, we leveraged on Androguard (https://androguard.readthedocs.io/), an open-source Android reverse engineering tool.

Given an Android APK, the DEX file D is a set of i Java compiled classes, $D = \{c_1 \ldots\ldots c_i\}$, where each class c contains j number of methods, $c = \{m_1 \ldots\ldots m_j\}$, with each method m consisting of a sequence of k assembly opcodes, $m = \{o_1 \ldots\ldots o_k\}$.

Multiple feature sets of n-gram opcode sequences as shown in Figure 2 were obtained after sliding a window of size n over the extracted assembly instruction code sequences. Due to the computational complexities associated with higher n-gram sizes, we limit the size of n to 2. Research including a recent study by Yeboah et al. [26] has shown how bigram features reinforces contextual relationship in opcodes sequences after yielding the best accuracies on their malware detection model.

4.2. Opcode Sequence Preprocessing. The length of opcode sequences generated from each Android APK differs from one app to another, therefore requiring unification. In this study, we limit an app's opcode sequence to a fixed length L, where L is a hyper-parameter. Here, the choice of L can impact the accuracy of the model. Ordinarily, higher L size could potentially yield better performance compared to smaller L size. To unify opcode sequence sizes, we truncate an app's opcode sequence of length l to L if the size is greater than L, otherwise if the sequence length is less than L (i.e., $l < L$), we pad a total of $L - l$ zeros to the sequence. With limited resources, we chose L to be 600.

Another issue we addressed is the issue of opcode frequency usage. We noticed that some common opcodes such as "invoke-direct," "move-result-object," and "new-instance" were used frequently, whereas others including "sget-short" and "rem-float" were rarely encountered. The reality of rare opcodes is that they may not be represented accurately when embedded into their respective vectors as described in section 4.3. To overcome this limitation, opcodes with frequency count less than three were represented with a single token.

The same preprocessing operations were also applied to the bigram opcode sequence feature set, where bigram opcode sequence length l was shortened to L or zeros padded to sequences whose length was shorter than L. Additionally, bigram opcodes which were rarely seen were replaced with a single token to reinforce embedding to continuous vector space.

4.3. Opcode Semantic Embeddings. The preprocessed opcode sequences need to be represented in formats befitting the proposed 1D CNN downstream model. This could be accomplished with one-hot encoding [27], where each opcode is represented with a high-dimensional sparse vector. However, such sparse high-dimensional vectors could be computationally costly for our downstream model, given that the number of unique opcodes in the dataset is high. The inability for one-hot vectors to capture semantic

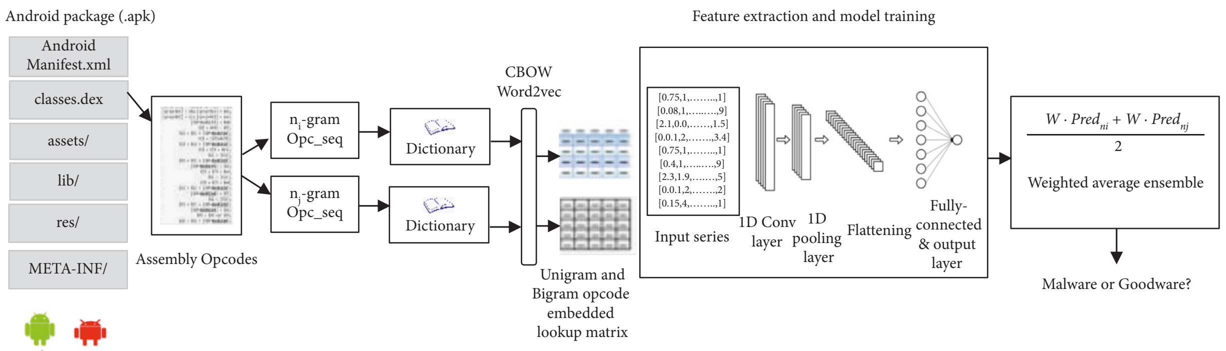
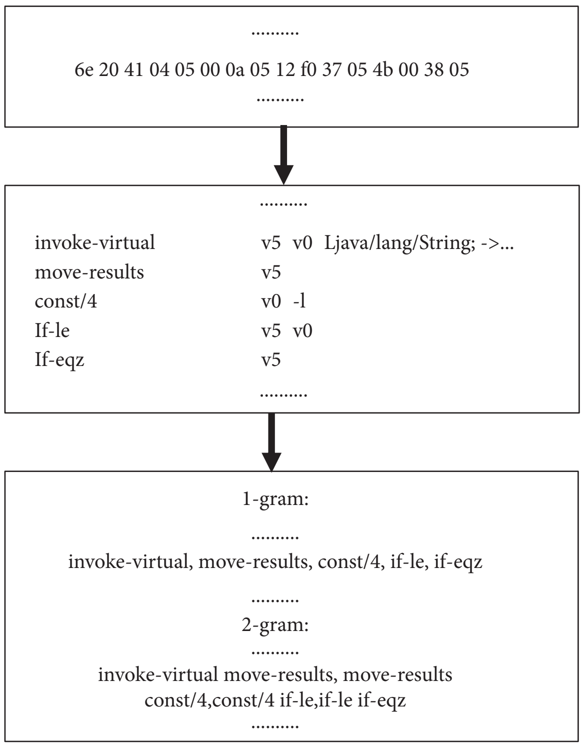


FIGURE 1: Proposed ensemble 1D convolutional neural network-based Android malware detection framework. ni-gram and nj-gram are unigram and bigram opcode sequences, respectively. Weights (W) are obtained using grid search on values between 0 and 1.



```
..........
6e 20 41 04 05 00 0a 05 12 f0 37 05 4b 00 38 05
..........

..........
invoke-virtual     v5  v0  Ljava/lang/String; ->...
move-results       v5
const/4            v0  -l
If-le              v5  v0
If-eqz             v5
..........

1-gram:
..........
invoke-virtual, move-results, const/4, if-le, if-eqz
..........
2-gram:
..........
invoke-virtual move-results, move-results
const/4,const/4 if-le,if-le if-eqz
..........
```

FIGURE 2: N-gram opcode sequence extraction.

relationships within opcode sequences also inhibits its usage, since semantic vector representations boost feature extraction with deep learning models. To overcome the highlighted issues surrounding one-hot vector representation, a more sophisticated embedding solution is required. Such an embedding solution should be capable of generating contextually related dense vectors, that is, opcodes which appear in similar context are represented with similar embedding vectors.

To represent opcodes in contextually dense real-valued vectors, we leverage on an approach proposed by Mikolov et al. [19], which learns words to vector embedding using a neural network. In 2013, Mikolov and a group of researchers in Google proposed the word2vec technique, which has since then been widely accepted and shown promising results on natural language processing-related tasks. Their article outlines two model architectures for the word2vec embedding: continuous bag-of-words (CBOW) model, which predicts a target word from surrounded context words, and skip-gram model, predicting context words from a given target word. While both word2vec models could be suitable for our studies, we chose CBOW which is found to train faster than skip-gram.

We trained the embedding model on both unigram and bigram opcode sequences separately and obtained an output of matrix $M \times N$ for each unigram and bigram feature sets, where M is the number of unique unigram or bigram opcodes and N is the dimension for each embedded opcode vector. Here, the size of M is dependent on the length of the input opcode sequences L, whereas the vector dimension N is a hyper-parameter. We chose N to be 64 for both unigram and bigram opcode embeddings.

4.4. Automated Feature Extraction and Model Training.

The application of deep neural networks (DNN) has become ubiquitous due to the state-of-the-art results achieved on real-world problems including natural language processing and computer vision-related tasks. One major benefit of deep learning models is the ability to automate feature extraction which over the years has been a daunting task to machine learning applications. CNNs have been widely adopted as the *de facto* deep learning model for 2-dimensional-related tasks like image processing. CNNs are regarded to be very efficient at detecting deep hidden patterns from huge volumes of visual data. Raw data input to a CNN is processed by first extracting low-level features using the convolutional layer via filter kernels, which are further subsampled to extract high-level features using the pooling layers.

Recently, convolutional neural networks have been applied to automated malware detection [21, 22, 24]. Most of such proposed CNN-based approaches to malware classification employ image processing techniques to create a 2D visual representation of static and dynamic features (mostly opcodes, API methods, or raw binary byte sequences) from executable files. Malicious and benign patterns are learned using 2D convolution filters, which scans the 2D binary representation left to right from top to bottom. While this approach has yielded great results, 2D convolution operations are generally computationally intensive. For example, a 2D input data nxn which convolves with a kernel of dimension kxk is bound to be executed at a complexity equivalent to $O\ (n^2k^2)$ computations.

1-dimensional convolutional neural network is another version of CNN models that has mostly been applied to time-series data. While 1D CNNs have also yielded state-of-the-art performance on real-word problems, the model comes at a computational advantage over 2D CNN. Given the above same n and k dimensions, computational complexities could be reduced to $O\ (nk)$ using 1D CNN. The major architectural difference between both versions of CNN models lies in the convolution and subsampling operations. Unlike 2D CNNs, 1D-based CNN models employ 1D convolutional filters and extract patterns by moving from the top of the input data to the bottom without moving to left or right as it does with 2D CNNs.

In this study, we leverage on 1D CNN to automate feature extraction on unigram and bigram opcode sequences. Here, we consider both feature sets as time-series data with length (opcode sequence length) L and width (opcode embedded vector dimension) N. The matrix $L \times N$ is a sequence of vectors used as input to the neural network. By this, convolution and subsampling operations are run in one direction from the top of the input to the bottom as shown in Figure 3. Results from both operations are flattened and fed into a fully connected layer where the actual verdict of a sample's maliciousness is decided. To prevent the model from overfitting, we added a regularization dropout layer. In Table 1, we show a summary of the end-to-end deep learning architecture which has been leveraged in this research for Android malware detection.

4.5. Fusion of Multi-Feature Sets via Weighted Average Ensemble. We employ boosting ensemble to fuse predictions from models trained on unigram and bigram opcode sequences. The benefit of using such a fusion approach is that the model prediction does not rely solely on a single trained feature set but instead combine predictions of models trained on multi-feature sets using a weighted average ensemble. This approach of ensemble according to recent studies [26, 29] is found to yield better performance in malware detection.

$$y = \arg\max\Big(\text{avg}\big(W \cdot \text{pred}_{ni}, W \cdot \text{pred}_{nj}\big)\Big). \tag{1}$$

As shown in (1), individual model predictions from trained unigram (n_i) and bigram (n_j) opcode feature sets are weighted, combined on average and passed through an argmax function [30] to give the final decision on the class (y) an executable file belongs. To obtain optimal weights (W), we employed a grid search on predefined values in the range 0 to 1.

5. Data Collection

The proposed model was evaluated on dataset collected from Canadian Institute of Cybersecurity (CIC) research Lab which comprises 4948 malware [31, 32] and 2477 benign [33] Android samples. The dataset had also been collected by CIC from sources including Contagio security blog, AMD, MalDozer, and VirusTotal. The malware is a component of 1595 SMS (randomly sampled), 1253 adware, and 2100 banking malicious applications.

6. Performance Metrics

Performance of the proposed deep 1D convolutional neural network model for Android malware detection has been evaluated on accuracy (Acc), which is the ratio of total correctly predicted observations to all predictions made as shown in (2); precision (Precc), which is the proportion of correct positive predictions to total positive predictions as shown in (3); recall (sensitivity), which is the actual detection rate measured by the ratio of positive predictions to all positive observations in the dataset as shown in (4); and F1-score, which is a score that factors in both precision and recall as shown in (5).

$$\text{Accuracy}\,(Acc) = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2}$$

$$\text{Precision}\,(\text{precc}) = \frac{TP}{TP + FP}, \tag{3}$$

$$\text{Recall} = \frac{TP}{TP + FN}, \tag{4}$$

$$F1 - \text{Score} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}, \tag{5}$$

where true negative (TN) is the correctly identified benign samples, true positive (TP) is the malware samples rightly detected, false positive (FP) is the wrongly detected malware samples, and false negative (FN) is the malware samples unable to be detected.

7. Experimental Results

We employed K-fold cross validation to evaluate the proposed deep 1D CNN framework. With K set to 5, the model was evaluated in 5 rounds of training and testing on 5 different subsets of our dataset. We computed global performance scores by averaging scores from all the rounds. In Table 2, we show experiment results of ensemble unigram and bigram Android opcode sequences. Results of nine experiments from different combination of weights (i.e., $W_{\text{1-gram}}$ and $W_{\text{2-gram}}$) shows that the 1D CNN malware detection model generally produced higher precision (Precc) scores compared to accuracy (Acc), revealing the model's sensitivity to true positive predictions. The discrepancies in accuracy and precision performances may be attributed to the imbalance in the number of malware and benign samples in the dataset.

In terms of precision, the best performance of 98.0% was obtained using an equal weight of 0.5 on ensemble unigram and bigram opcode sequences, whereas weight pair 0.2 and 0.8 yielded 95.5% as best accuracy on ensemble unigram and bigram sequences, respectively. The receiver operating characteristic curve (ROC) shown in Figure 4 yielded an

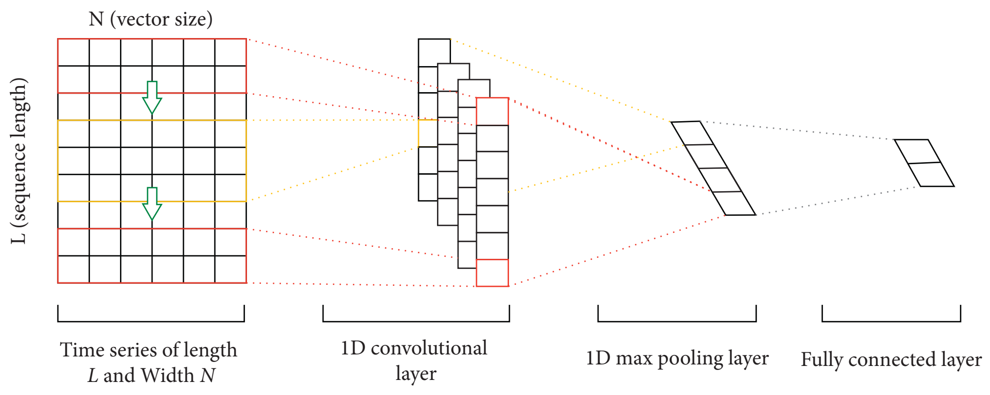


Figure 3: 1D convolution for multivariate time series Boyle [28].

Table 1: Neural network architecture for Android malware detection.

Layers	Parameters	Nonlinear activation
1D convolutionx3	Filters = 32, kernel_size = 3	ReLU
Regularization	Dropout = 0.5	ReLU
1D MaxPooling	Size = 3	ReLU
FC dense	No_neurons = 256	ReLU
FC dense	No_neurons = 1	Softmax

^{X3} denotes three convolution operations using the same parameters and activation function.

Table 2: Experimental results of ensemble unigram and bigram.

$W_{1\text{-gram}}$	$W_{2\text{-gram}}$	Accuracy	Precision
0.1	0.9	0.951	0.968
0.2	0.8	**0.955**	0.969
0.3	0.7	0.952	0.972
0.4	0.6	0.943	0.978
0.5	0.5	0.934	**0.980**
0.6	0.4	0.930	0.956
0.7	0.3	0.823	0.978
0.8	0.2	0.544	0.932
0.9	0.1	0.562	0.791

average area under curve (AUC) score of 0.98 for the best ensemble model in terms of precision.

Opcode sequences using different combination of grid search weights.

Classification report for the best ensemble model in terms of precision is shown in Table 3. Similarly, Tables 4 and 5 also show performances of the 1D CNN model trained on individual n-gram feature sets. We observed that the ensemble model trained with a parity weight of 0.5 on unigram and bigram opcode sequences outperformed the model trained on a single n-gram feature set. The ensemble model yielded an average precision score of 98%, with recall and F1-score values of 97%. These performances in precision, recall, and F1-score happens to be better than the single trained n-gram models shown in Tables 4 and 5. The bigram trained model also performed slightly better on precision than the model trained on unigram opcode sequences.

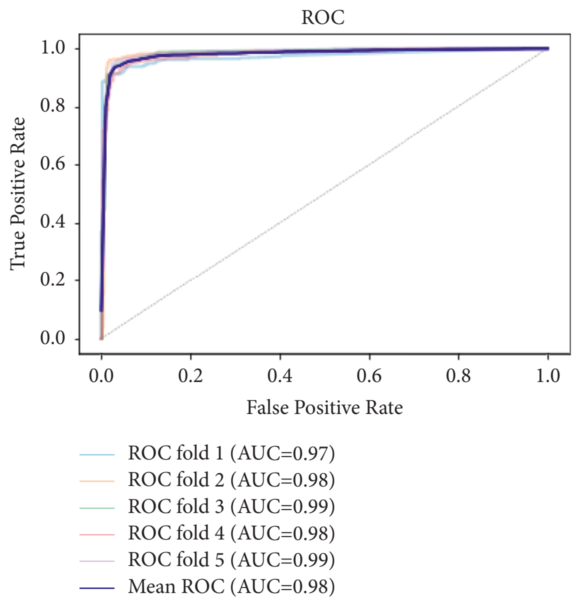


Figure 4: Receiver operating characteristic curve for ensemble model trained with parity weights.

Table 3: Classification report of best (in terms of precision) ensemble unigram and bigram opcode sequences..

	Precision	Recall	F1-score	Support
Goodware	0.96	0.95	0.95	2477
Malware	0.99	0.98	0.98	4948
Average	0.98	0.97	0.97	7425

7.1. Comparative Classification Results. In this section, we compare performance of our proposed framework to three peer technique malware detection frameworks proposed by Aksakalli [34], Jung et al. [35], and Hasegawa and Iyatomi [36]. Both [34, 35] employed 2-dimensional CNN for feature extraction on Android executables, as both methods are most likely to be computationally intensive than the method proposed in this work. Hasegawa and Iyatomi [36], on the other hand, proposes a lightweight model based on 1-dimensional

Table 4: Classification report of Unigram opcode sequence feature set.

	Precision	Recall	F1-score	Support
Goodware	0.91	0.94	0.93	2477
Malware	0.97	0.95	0.96	4948
Average	0.94	0.95	0.95	7425

Table 5: Classification report of bigram opcode sequence feature set.

	Precision	Recall	F1-score	Support
Goodware	0.93	0.93	0.93	2477
Malware	0.97	0.96	0.97	4948
Average	0.95	0.95	0.95	7425

Table 6: Comparative analysis with peer technique models.

	Architecture	Dataset (no.)	Features	Feature representation	Results (%)
Jung et al. [35]	Inception-v3 and Inception-ResNet-v2 (2D CNN)	Malware: 5377 Benign: 6249	Data section bytes	Grayscale image	Acc: 98.02
Aksakalli [34]	2D CNN	Malware: 2500 Benign: 2500	Permissions	One-hot sparse vector	Acc: 96.71
Hasegawa and Iyatomi [36]	1D CNN	Malware: 5000 Benign: 2000	Raw APK bytes	Byte representation	Acc: 97.04
Our model	Ensemble 1D CNN	Malware: 4948 Benign: 2477	n-gram static opcodes	Word2vec embedding	Acc: 95.5 Precc: 98.0

CNN. All three techniques are evaluated on a moderate-sized malware dataset, where [34–36] have been evaluated on 2500, 5377, and 5000 malware samples, respectively.

Jung et al. [35] applied image techniques for malware representation and further trained two state-of-the-art CNN models namely Inception-v3 and Inception-ResNet-v2. The Inception-ResNet-v2 model was able to detect malware at an accuracy of 98.02%. Aksakalli [34] also learns malicious and benign patterns from the app's permissions. Their architecture, which comprises a convolutional neural network with few layers, also recorded a detection accuracy of 96.71%. Hasegawa and Iyatomi [36] trained a 1D CNN network on the last 512-1K raw bytes sequence and saw 97.04% accuracy using few convolutional layers. The promising results of our computationally efficient model as shown in Table 6 reveal our solution could be deployed to augment or possibly substitute other CNN-based solutions, especially on low-end devices. One other advantage our model has over the other three peer models is that, in terms of susceptibility to adversarial obfuscation, our model is likely to perform better as mentioned earlier in section 3.

8. Limitations

The performance of machine learning-based solutions face two major threats, which our model is no exception of. First, the model may suffer detection downgrade as malware evolves over time, a threat described as *concept drift*. The proposed model is likely to be obsolete as features used in training the model drifts over time. Hence the resilience of our model against evolution needs to be looked at. Although recent works are seen to propose solutions to tackle concept drift [37], this work does not factor in such measures, therefore we entreat a constant update upon deployment.

As reported in [38, 39], *adversarial evasion* is another common machine learning threat our model is vulnerable to. Evasion by adversarial tactics aims at creating new versions of samples known as adversarial examples, which are intended to induce a model to misclassification at test time. Authors of [39] have shown that machine learning-based image classification models could easily be bypassed by the insertion of small perturbations of pixels into an input image leading to classification errors. Such adversarial image obfuscation approaches when applied to malware are inhibited by the semantic nature of executables samples. Binary files which come packed in predefined formats are characterized by interdependence between proximate bytes and hence an attempt to arbitrarily insert bytes will likely alter the executable's operation or at worse cause the executable not to run at all. Nonetheless, the works by [38, 40] show machine learning-based malware detectors are potential candidates for adversarial attacks. Our model needs to be further studied in this domain to evaluate resilience against adversarial evasions.

9. Conclusion

In this study, we have proposed and evaluated a computationally efficient 1-dimensional convolutional neural network to automate Android malware detection. Raw static opcode sequences are first extracted from Android binaries, followed by a preprocessing stage where sequence lengths are unified and least seen opcodes replaced with a single token. Subsequently, word-2vec is employed to pretrain n-gram opcode sequence feature sets to generate semantic-preserved vector representations. Further, a densely connected 1D convolutional neural network is trained on unigram and bigram opcode sequence feature sets using the word2vec pretrained vector representations. Experimental results have shown that the fusion of predictions from the trained unigram and bigram opcode feature sets enhances malware detection on the Android dataset. With precision and detection rate reaching 98% and 97%, respectively, the ensemble unigram and bigram opcode sequences outperformed the model trained on individual n-gram feature sets.

For future work, we would like to go beyond bigram opcode sequences and test the model on higher n-grams. Additionally, we shall explore other semantic embedding techniques for opcode sequence representation. Lastly, resilience against adversarial attacks will also be investigated.

Data Availability

The dataset that has been leveraged to evaluate our proposed model is publicly available in the following repository: https://doi.org/10.6084/m9.figshare.18865805.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] S. O’Dea, “Smartphones – Statistics and Facts,” 2022, https://www.statista.com/topics/840/smartphones/#dossierKeyfigures.

[2] Lookout Energy, “Lookout Energy Industry Threat Report: Mobile Phishing Threats Surged 161% in 2021,” 2022, https://www.lookout.com/news-release/lookout-energy-report-shows-mobile-phishing-surged-161-percent.

[3] E. Konstantinou, *Metamorphic Virus: Analysis and Detection*, Technical Report RHUL-MA, India, 2008.

[4] M. R. Chouchane and A. Lakhotia, “Using Engine Signature to Detect Metamorphic malware,” in *Proceedings of the 4th ACM workshop on Recurring malcode - WORM ’06*, pp. 73–78, ACM, New York, NY, USA, July 2006.

[5] D. Gibert, C. Mateu, and J. Planes, “The rise of machine learning for detection and classification of malware: research developments, trends and challenges,” *Journal of Network and Computer Applications*, vol. 153, pp. 102526–102620, 2020.

[6] K. N. Khan, M. s. Khan, M. Nauman, and M. Y. Khan, “Op2Vec: An Opcode Embedding Technique and Dataset Design for End-To-End Detection of Android Malware,” *Security and Communication Networks*, vol. 2021, 2021.

[7] A. O. Christianah, B. A. Gyunka, and A. N. Oluwatobi, “Optimizing android malware detection via ensemble learning,” *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 14, no. 9, pp. 61–74, 2020.

[8] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. R. Drebin, “Effective and explainable detection of android malware in your pocket,” in *Network and Distributed System Security Symposium*, pp. 1–15, Sidhananda Nagar, Puducherry, 2014.

[9] Y. Aafer, W. Du, and H. Yin, “DroidAPIMiner: mining API-level features for robust malware detection in android,” in *International Conference on Security and Privacy in Communication Systems*, pp. 86–103, Sydney, Australia, 2013.

[10] A. Desnos, G. Gueguen, and B. Androguard, https://androguard.readthedocs.io/en/latest/, 2022.

[11] APKTool, “A Tool for Reverse Engineering Android Apk Files,” 2022, https://ibotpeaches.github.io/Apktool/.

[12] J. Freke, “An Assembler/disassembler for the Dex Format,” 2022, https://github.com/JesusFreke/smali.

[13] S. K. Dash, G. Suarez-Tangil, S. Khan et al., “DroidScribe: Classifying Android Malware Based on Runtime Behavior,” in *Proceedings of the 2016 IEEE Security and Privacy Workshops (SPW)*, pp. 252–261, IEEE, San Jose, CA, USA, May 2016.

[14] M. Dimjašević, S. Atzeni, I. Ugrina, and Z. Rakamaric, “Evaluation of android malware detection based on system calls,” in *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*, pp. 1–8, Louisiana, New Orleans, USA, March 11, 2016.

[15] Y. Zhou, Z. Wang, W. Zhou, and Y. JiangHey, “Get off of my market: detecting malicious apps in official and alternative android markets,” in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, Sidhananda Nagar Puducherry, 2012.

[16] W. Enck, P. Gilbert, B. gon Chun et al., “An information-flow tracking system for realtime privacy monitoring on smartphones,” *Symposium on Operating Systems Design and Implementation*, pp. 393–407, OSDI), Ottawa, Montreal, Toronto, 2010.

[17] L. K. Yan and H. Y. Droidscope, “Seamlessly reconstructing OS and Dalvik semantic views for dynamic android malware analysis,” *Proc. of USENIX Security Symposium*, vol. 29, 2012.

[18] P. Huilgol, “Quick Introduction to Bag-Of-Words (BoW) and TF-IDF for Creating Features from Text,” 2022, https://www.analyticsvidhya.com/blog/2020/02/quick-introduction-bag-of-words-bow-tf-idf/.

[19] T. Mikolov, G. Corrado, K. Chen, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” pp. 1–12, 2013, https://arxiv.org/abs/1301.3781.

[20] J. Pennington, R. Socher, and C. D. M. GloVe, “Global vectors for word representation,” in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 1532–1543, EMNLP), CA, San Francisco, USA, May 13 - 17, 2019.

[21] E. B. Karbab and M. Debbabi, “MalDy: p,” *Digital Investigation*, vol. 28, pp. S77–S87, 2019.

[22] J. Sun, X. Luo, H. Gao, W. Wang, Y. Gao, and X. Yang, “Categorizing malware via A Word2Vec-based temporal convolutional network scheme,” *Journal of Cloud Computing*, vol. 9, no. 1, pp. 53–14, 2020.

[23] A. Sharma, P. Malacaria, and M. Khouzani, “Malware Detection Using 1-Dimensional Convolutional Neural Networks,” in *Proceedings of the 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), IEEE, Stockholm, Sweden*, 17-19 June 2019.

[24] J. Hemalatha, S. Roseline, S. Geetha, S. Kadry, and R. Damaševičius, “An efficient DenseNet-based deep learning model for malware detection,” *Entropy*, vol. 23, no. 3, pp. 1–23, 2021.

[25] T. Strauss, M. Hanselmann, A. Junginger, and H. Ulmer, “Ensemble Methods as a Defense to Adversarial Perturbations against Deep Learning,” CoRR, 2017, http://arxiv.org/abs/1709.03423.

[26] P. N. Yeboah, S. K. Amuquandoh, and H. B. B. Musah, “Malware detection using ensemble N-gram opcode sequences,” *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 15, no. 24, pp. 19–31, 2021.

[27] K. Potdar, “A comparative study of categorical variable encoding techniques for neural network classifiers,” *International Journal of Computer Application*, vol. 175, no. 4, pp. 7–9, 2017.

[28] M. Boyle, “1-D Convolution for Time Series,” 2021, https://morioh.com/p/6a0e45b9f06f.

[29] A. Martín, R. Lara-Cabrera, and D. Camacho, “Android malware detection through hybrid features fusion and ensemble classifiers: the AndroPyTool framework and the OmniDroid dataset,” *Information Fusion*, vol. 52, pp. 128–142, 2019.

[30] SciPy org, “SciPy.org,” 2021, https://docs.scipy.org/doc/numpy-1.9.1/reference/generated/numpy.ma.MaskedArray.argmax.html.

[31] S. Mahdavifar, A. F. A. Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, “Dynamic android malware category classification using semi-supervised deep learning,” in *Proceedings of the 18th IEEE International Conference on Dependable, Autonomic, and Secure Computing (DASC)*, Calgary, AB, Canada, 17-22 Aug. 2020.

[32] 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, pp. 22–34, 2022.

[33] L. Taheri, A. F. Abdulkadir, and A. H. Lashkari, “Extensible android malware detection and family classification using network-flows and API-calls,” in *Proceedings of the IEEE (53rd) International Carnahan Conference on Security Technology*, Chennai, India, 1-3 Oct. 2019.

[34] I. K. Aksakalli, “Using convolutional neural network for android malware detection,” *Computer Modelling and New Technologies*, vol. 23, no. 1, pp. 29–35, 2019.

[35] J. Jung, J. Choi, S. J. Cho, S. Han, and M. Park, “Android Malware Detection Using Convolutional Neural Networks and Data Section Images,” in *Proceedings of the 2018 Conference on Research in Adaptive and Convergent*, Hawaii, Honolulu, October 9 - 12, 2018.

[36] C. Hasegawa and H. Iyatomi, “One-dimensional convolutional neural networks for android malware detection,” in *Proceedings of the 2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*, pp. 101–104, Penang, Malaysia, 9-10 March 2018.

[37] M. Hashmani, M. J. Syed, M. Rehman, and A. Inoue, “Concept drift evolution in machine learning approaches: a systematic literature review,” *International Journal on Smart Sensing and Intelligent Systems*, vol. 13, 2020.

[38] F. Kreuk, A. Barak, S. Aviv, M. Baruch, B. Pinkas, and J. Keshet, “Deceiving End-To-End Deep Learning Malware Detectors Using Adversarial Examples,” pp. 1–6, 2019, https://arxiv.org/abs/1802.04528.

[39] I. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and Harnessing Adversarial Examples,” 2014, https://arxiv.org/abs/1412.6572.

[40] O. Suciu, S. E. Coull, and J. Johns, “Exploring adversarial examples in malware detection,” in *Proceedings of the 2019 IEEE Security and Privacy Workshops (SPW), IEEE, San Francisco, CA, USA*, 19-23 May 2019.