Journal of Database Management
Volume 34 • Issue 2

Android-IoT Malware Classification and Detection Approach Using Deep URL Features Analysis

Farhan Ullah, School of Software, Northwestern Polytechnical University, Xi'an, China*
https://orcid.org/0000-0002-1030-1275

Xiaochun Cheng, Department of Computer Science, Swansea University, Bay Campus, Fabian Way, Swansea, UK

Leonardo Mostarda, Computer Science Department, Camerino University, Camerino, Italy
https://orcid.org/0000-0001-8852-8317

Sohail Jabbar, Department of Computational Science, The University of Faisalabad, Faisalabad, Pakistan

ABSTRACT

Currently, malware attacks pose a high risk to compromise the security of Android-IoT apps. These threats have the potential to steal critical information, causing economic, social, and financial harm. Because of their constant availability on the network, Android apps are easily attacked by URL-based traffic. In this paper, an Android malware classification and detection approach using deep and broad URL feature mining is proposed. This study entails the development of a novel traffic data preprocessing and transformation method that can detect malicious apps using network traffic analysis. The encrypted URL-based traffic is mined to decrypt the transmitted data. To extract the sequenced features, the N-gram analysis method is used, and afterward, the singular value decomposition (SVD) method is utilized to reduce the features while preserving the actual semantics. The latent features are extracted using the latent semantic analysis tool. Finally, CNN-LSTM, a multi-view deep learning approach, is designed for effective malware classification and detection.

KEYWORDS

Android, Deep Learning, HTTP flow analysis, Internet of Things, LSA, Malware detection, N-gram, Text Semantics

1. INTRODUCTION

A malware infection can easily attack Android apps for malicious purposes and compromise security (Lu & Da Xu, 2018). Mobile network expansion has increased the number of portable devices. Because of this, financial malware apps threaten mobile users. Despite massive prevention and mitigation efforts, malware remains a major cyber security threat. Thus, in 2016, Symantec

DOI: 10.4018/JDM.318414
*Corresponding Author

discovered 357,019,453, in 2017, 669,974,865, and in 2018, 246,002,762 new malware variants. Yet more malware variants are attempting to bypass anti-virus tools and avoid detection by several malware detection systems.

The rapid expansion of mobile interaction places a significant burden on smartphone security management. According to a recent study[1,] the number of apps in the Google Play Store has increased from 16K in Dec. 2009 to over 2 million in Feb. 2016. As a result, mobile traffic has topped 3.7 exabytes. The growth of the mobile ecosystem is seriously compromised by malicious apps. There has been a massive increase in mobile malware, especially targeting Android devices. Devastating digital payment thefts and other attacks threaten mobile security. Despite the Android platforms and mobile antivirus security measures, sophisticated mobile malware continues to infiltrate mobile systems. The widespread use of mobile devices also exposes users to multiple risks. So we urgently need Android-based mobile malware detection systems (McLaughlin et al., 2017). A malware family is a group of malicious apps sharing code. Various malware samples use the malware families' codebase. All samples with the same interpretation are combined.

Faruki et al. (Faruki et al., 2014) explore the characteristics of a huge assortment of malware and categorizes existing mobile malware detection methods into static, dynamic, and traffic-based categories. Static analysis has been used in several previous studies to discover data leakage, malware, and security breaches in Android apps (Zhu et al., 2018). Nevertheless, static analysis of malware is challenging due to code polymorphism and obfuscation. These methods are used to produce malware variants to avoid detection. Numerous different dynamic analysis techniques strive to alter the device's operating system to monitor and access confidential information at runtime (Bader, Lichy, Hajaj, Dubin, & Dvir, 2022; Ucci, Aniello, & Baldoni, 2019). Such methods are helpful, but they necessitate a massive number of executions to encompass all app behavioral patterns (Ahmed, Lin, & Srivastava, 2021).

1.1 Motivation

Many virus detection techniques concentrate on the network traffic generated by mobile apps. Malware is identified by abnormal network behavior patterns. This type of malware detection system has the potential to be effective because the majority of Android malware performs its malicious functions via network traffic (Zhou & Jiang, 2012). The malware must communicate with a remote server over the Internet to carry out malicious tasks. These traces can be used to identify and track down specific malware. Furthermore, malware detection strategies based on network characteristics are more straightforward to design and implement than static or dynamic analytic approaches. For example, methods based on traffic detection can be installed at an access point or gateway. These methods rely solely on user-generated network traffic data, ensuring that users do not lose access to their mobile resources. Furthermore, these solutions do not necessitate any user actions aside from granting licenses to the detection service (W. Li, Bao, Zhang, & Li, 2022; S. Wang et al., 2020). The goal of network traffic-based approaches is to find distinguishing features that can be used to classify malware more effectively. Selecting efficient features, on the other hand, is a difficult task. We concentrate our investigation on malware samples that use the HTTP/HTTPS protocol to send data. Because HTTP accounts for 70% of the network traffic generated by Android apps, we chose it for our research. (Dai, Tongaonkar, Wang, Nucci, & Song, 2013). However, because HTTP traffic is generated in encrypted form, extracting useful information from it is extremely difficult.

1.2 Contributions

Previously, researchers used statistical features and network traffic content to deduce malware behavior, which can be tedious and inefficient. Therefore, we develop a method for detecting malware using malicious URLs (HTTP/HTTPS). It does not require a lengthy and complex feature selection procedure. To accomplish the automatic feature selection, we employ a novel features extraction process (Mikolov, Chen, Corrado, & Dean, 2013; S. Wang et al., 2017a). This method can automatically mine URL feature representations at numerous levels to tackle the problem of selecting features while preserving the high-level semantics. The main contributions of the paper are given as follows:

1. On the URLs in network traffic, we conduct feature extraction and selection using a text-based approach. Our research involves the development of a novel traffic data preprocessing and transformation method that is effective in the detection of malware using network flows.
2. The N-gram features analysis approach is used to mine the features' sequencing and then the Singular Value Decomposition (SVD) method is applied for features reduction preserving the actual semantics. The Latent Semantic Analysis (LSA) tool is employed to extract the latent features.
3. To extract deep and broad feature learning, we design a multi-view deep learning approach called Convolution Neural Network-Long Short Term Memory (CNN-LSTM). The novel approach is capable of producing discriminant features and thus mitigates the difficulty associated with feature selection in network traffic-based malware detection methods.

The remaining sections of the paper are as follows: Section 2 describes recent and related literature, Section 3 describes the proposed approach, Section 4 describes the results and discussions, and Section 5 concludes.

2. BACKGROUND

Several studies (Chiramdasu, Srivastava, Bhattacharya, Reddy, & Gadekallu, 2021; Varma, Raj, & Raju, 2017) show that the Android platform employs a variety of security measures, including authorization mechanisms to protect infecting target devices. But users must be sufficiently aware of security issues to benefit from authorization-based protection. These constraints of over-reliance on the consumer make Android malware invade and spread through mobile devices. Most of these scanners look at things like authorizations and harmful programs to see if an app is malicious or not. These anti-virus scanners protect against harmful software. But malicious programs are always evolving and diversifying. So malware detection must be improved. There are now several malware detection tools that can dissect APK files without executing them. Sanz et al. (Sanz et al., 2013) proposed a static-based analysis that captures the app's uses-permission and uses-feature details for malware classification, with an accuracy of 86.41%.

Shanshan et al. (S. Wang et al., 2017a) suggested using network traffic as semantic content to detect malware. A mobile app's HTTP flow is a text file. Natural Language Processing (NLP) can extract semantics from HTTP traffic data stored in text files. The next step is to detect malware using network traffic textual properties. The suggested approach obtains the classification performance with an accuracy rate of 99%. Aresu et al. (Aresu, Ariu, Ahmadi, Maiorca, & Giacinto, 2015), look at HTTP traffic when Android malware communicates with distant malicious servers. It also uses clustering to build malware family signatures. This signature is used to detect malicious attacks. Wang et al. (S. Wang et al., 2017b), proposed the TextDroid approach that divides the HTTP packet's content by special characters, and after that, generates n-gram sequence information to study the ordering in the collected features. Further, it collected sequence-based information fed into a machine-learning model for malware detection. However, the detection performance for this semantic-based approach is only 76.99%. Shyong et al. (Shyong, Jeng, & Chen, 2020), identified Android apps using both static and dynamic network monitoring. Dynamic analysis identifies malware families by analyzing network data. The static technique classified valid and harmful content at 98.86%. Also, the dynamic app family classification is 96% accurate.

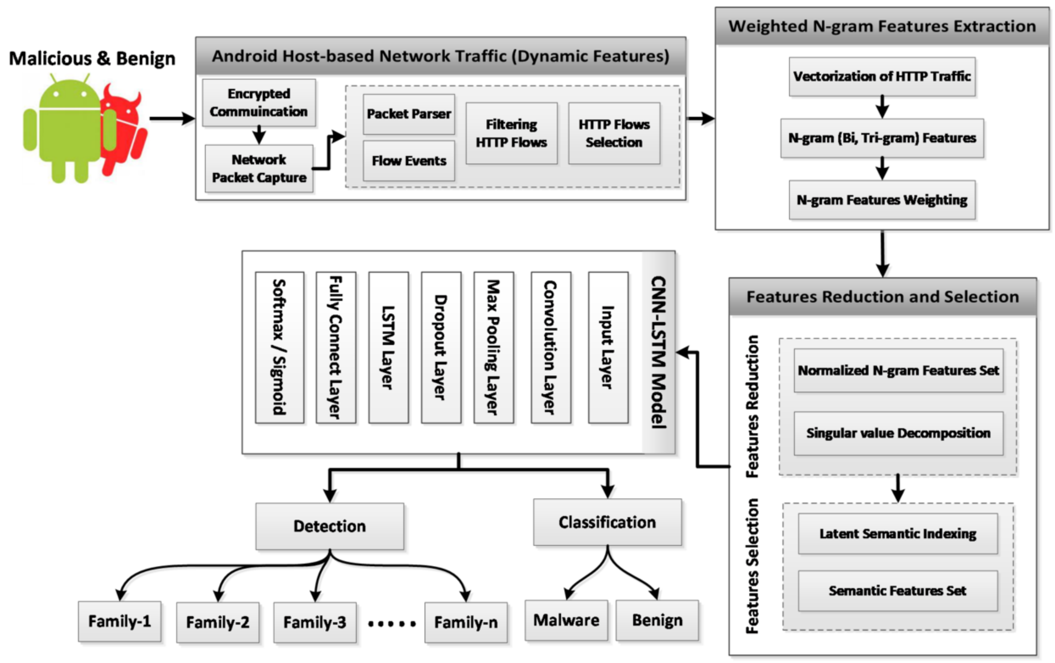
Several deep learning-based malware classification studies have attained outstanding results (S. Wang et al., 2018; Xu, Eckert, & Zarras, 2021). Chen et al. (Chen, Yu, Zhang, Zhang, & Xu, 2016), proposed a convolutional neural network (CNN) method that uses HTTP headers to identify different types of apps based on their generated traces. The use of CNN speeds up the process of selecting features and produces precise results in traffic identification. The given method achieved an average classification accuracy of 98%. David et al. (David & Netanyahu, 2015) presented a deep belief networks approach known as DeepSign. It can produce immutable compact interpretations of malware actions, possibly allowing it to distinguish nearly all current malware variants efficiently with an accuracy rate of 98.6%. Shanshan et al. (S. Wang et al., 2020), presented a malware detection system based on app HTTP visits. Then a multi-view neural network detects malicious acts with depth. This method can focus smooth attention on specific input variable traits. This method's accuracy rates are 98.81% and 89.3%, respectively. Brandon et al. (Laughlin, Sankaranarayanan, & El-Khatib, 2020) describe a service plate form for delivering contextual data obtained from network flow. It creates a set of contexts for network-based features such as host addresses, apps in use, and the time of the event. The system finds the nearest neighbors in each context, evaluates the feature proportions, and employs an ensemble of the unsupervised outlier detection algorithm. The proposed method has an accuracy of 85.56%.

We use deep features analysis to process HTTP network analysis because it is the most widely used protocol for mobile apps. HTTP-based encrypted data packets are collected and filtered using a features extraction process, and then the SVD mathematical model is used to extract the most significant features in a reduced dimensional space. CNN-LSTM, a multi-view deep learning strategy, is developed to obtain extensive feature learning.

3. ANDROID MALWARE CLASSIFICATION AND DETECTION APPROACH

Figure 1 depicts the proposed method. The Android HTTP traffic is collected and fed into the malware detection system. Because the data is encrypted, a packet parser is required to decrypt it. Then, n-gram analysis is used to extract features with ordering details, and TFIDF is used to mine each feature's importance. The approach also aims to extract meaningful features in a reduced dimensional space.

Figure 1. Proposed Framework for Android Malware Classification and Detection using URLs



The reduced features are used in a CNN-LSTM model for malware classification and detection. Examples of malware families are shown in Figure 2. Each sample has its way of executing malicious Android apps. For instance, Adware is a type of computer program that displays ads on screen while using a web browser. Similarly, a botnet can be used to launch Distributed Denial of Service (DDoS) attacks, steal information, spear phishing, and allow hackers to obtain access to the device (Rieck, Holz, Willems, Düssel, & Laskov, 2008).

3.1 URL Collection

An app can be infected with malware after visiting a malicious URL. Malware frequently uses URL information to receive inputs and conduct suspicious attacks. So, detecting malware via URLs is useful. Android devices collect HTTP-based network data while online. The data is encrypted and may contain malicious or benign activities. Wireshark, a packet parser, decrypts network packets so that Android apps can explore network flows. It contains a lot of noisy data after decryption, which could affect malware classification accuracy. We use Wireshark's packet filtering command to mine the URLs because we need to focus on HTTP-based network traces. In addition, in the next step, we mined HTTP and HTTPS network traffic for features analysis (Narudin, Feizollah, Anuar, & Gani, 2016).

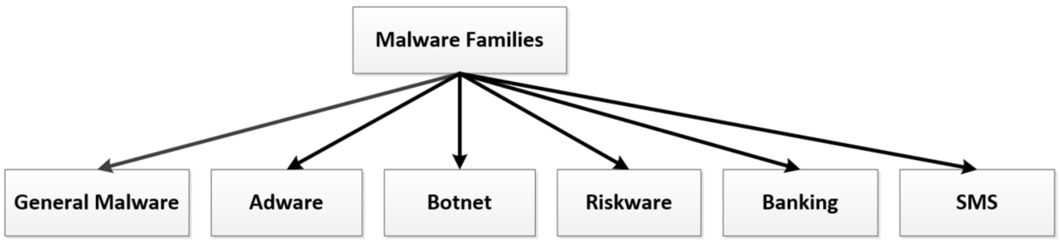
3.2 URL Preprocessing and Features Extraction

The network flow is a fundamental unit in malicious traffic detection. However, many app-generated HTTP flows are mixed. So, we must extract each comprehensive flow from the complex traffic file and segment them.

3.2.1 Traffic Flow Segmentation

Processing the URL into useful segments is difficult because standard tokens (like spacing or punctuation) are not available. Each URL contains a long string of characters, none of which can convey complex information. For instance, the extraction of a single character from the URL "www.yahoo.com" can be illogical. This can only exhibit a full domain name is considered as an entity. Another example of a video URL, i.e. "https://www.youtube.com/watch?v=-JAYqHNx5rU" is composed of some common and special characters. These prefixes are called HTTP flow headers. They show the method, encoding type, URL, and browser information in a coherent way. Each URL has basic segments that convey a specific piece of information. Because not all segmented terms can detect malware, we remove meaningless words from the traffic flow. This reduces computational costs. We develop filtering rules to remove noisy data. First, the noisy and low-frequency terms (".jpg", ".png", ".gif", ".js", ".css", etc.) are removed. Second, common terms like "content-length", "expires", and "en-us" is removed from almost every HTTP request. Third, stop words ("is", "are", "the", etc.) are removed as they can cause false positives in malware detection (S. Wang et al., 2017a). These terms may not be helpful in the malware detection system.

Figure 2. Android Malware Families



3.2.2 N-gram Analysis

In NLP, an N-gram is a random string of words. An N-gram can be used as a sub-sequence for elements in a specific order with at least n elements. Flow segmentation provides an N-gram ordering for each term set. For example, the term set of HTTP flow in uni-gram form is "apikey airpush appid 60563 longitude 0" The flow of bigram can be "apikey-airpush, airpush-appid, appid-60563, 60563-longitude, and so on". The flow of trigram can be "apikey-airpush-appid, airpush-appid-60563, etc." The N-gram can be used to find important term connections in a network flow. "apikey" and "airpush" are 1-gram segments that not only define a term but also show that they are unrelated. The bi-gram sequence "apikey-airpush" is influenced by the term "apikey". The N value chosen must appropriately reflect the principle of term presence in the HTTP flow header. TFIDF (Karbab & Debbabi, 2019) is another method for determining the relative importance of different N-gram features. The local and global weights show the significance of each N-gram feature in a single or multiple network flow, respectively. Mathematically, TFIDF is given as in equation 1.

$$\mathrm{tfidf}\left(\mathrm{t},\mathrm{d},\mathrm{D}\right)=\mathrm{tf}\left(\mathrm{t},\mathrm{d}\right).\mathrm{idf}\left(\mathrm{t},\mathrm{D}\right) \tag{1}$$

Where t, f, d, and D denote term, frequency, single network flow, and multiple networks flow, respectively. These parameters can extract the local and global weights collectively. Equation 2 shows how many times each N-gram feature is referenced.

$$tf\left(t,d\right)=\frac{f_{t,f}}{\sum_{t'\in d} f_{t',d}} \tag{2}$$

equation 3 illustrates logTF function.

$$\mathrm{LogTF}=\log\left(1+\mathrm{tf}\right) \tag{3}$$

The inverse document frequency of each feature's local weight is derived from equation 4 using the normalized term frequency form.

$$\mathrm{idf}\left(\mathrm{t},\mathrm{D}\right)=\log\frac{\mathrm{N}}{\left|\left\{\mathrm{d}\in\mathrm{D}:\mathrm{t}\in\mathrm{d}\right\}\right|} \tag{4}$$

Where t, d, D, and N show terms, single network flow, multiple networks flow in a group, and all networks flow, respectively. The TFIDF works in two main steps, such as selecting the source code document and then indexing it to a specific address for extracting the TF*IDF value. The indexing process takes time O(n), several times the source code document chosen is O(|D|). Therefore, the total computational time of the TFIDF method is O(|D| n).

3.3 Features Reduction and Selection

3.3.1 Features Reduction

We need enough records to effectively train a deep learning model. Less spatially relevant data requires semantic connections based on LSA. It uses a discrete representation space to retrieve semantic content from text documents. In Latent Semantic Indexing (LSI), latent constructs are used to find semantic-based concepts. It divides N-gram features into matrices that can classify malicious traffic (Hofmann, 2013; Landauer, McNamara, Dennis, & Kintsch, 2013). A dense N-gram feature is reduced using the SVD algorithm. To find patterns across networks, we need to use semantic factors that accurately explain network traffic. The LSA method can derive latent variables from network flows. The SVD algorithm converts N-gram features into three matrices (Landauer et al., 2013), i.e. M, U, Σ, V^{T} as shown in equation 5.

$$M = U\Sigma V^{T} \tag{5}$$

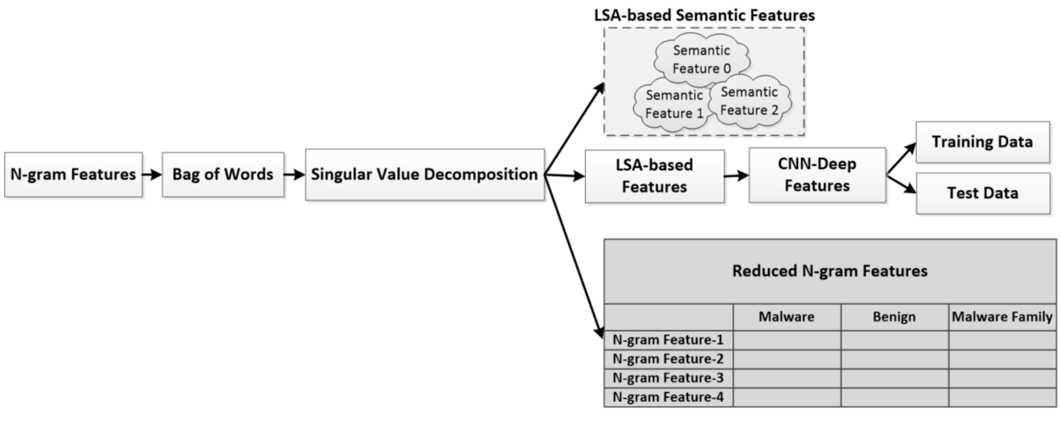
Where M is $\mathrm{m*n}$ matrix, U is an $\mathrm{m*n}$ singular matrix, Σ is $\mathrm{n*n}$ diagonal matrix, V^{T} is the transpose of the V matrix. N-gram features are fed into the SVD algorithm employing the Bag of Words (BoW) model as depicted in Figure 3. In the SVD algorithm, features are reduced from a large number to a small number of meaningful ones. Each N-gram feature is categorized as either malware, benign, or a member of a specific family of malware. By using the LSA method, the latent features can then be extracted from their original location in three-dimensional space.

The time complexity of LSA is given as follows. The LSA mainly works on three main matrices, such as M, U, Σ, V^{T}. These matrices are already discussed above with detailed information. The computational time is the multiplication of these matrices, which is $O(M\ U\ \Sigma\ V^{T})$.

3.3.2 Deep Features Selection

Though, the LSA-based features are highly effective but we still need deep features which can decrease the training cost of the proposed deep learning model. The CNN network is used to examine the reduced features and further mine the deep features. There are several studies available (Lee, Saxe, & Harang, 2019; Vasan, Alazab, Wassan, Safaei, & Zheng, 2020), that used the CNN model to classify malware features. The CNN model works better with a variety of data, like text, images, and videos. As presented in Figure 4, the approach can be adopted of a one-dimensional CNN network with convolutional layers, pooling layers, dropout layers, and a fully connected layer. The LSA-based features are input sequences to the CNN network. Tensor flow library 1.9 is used to build the CNN network. By iteratively spinning through the semantic features, convolution filters out the best deep features. Each filter creates a new feature map. The number of filters is determined by hyperparameter optimization. We used two CNN layers with 64 and 128 filters each. The max-pooling layer reduces the feature space's size, range, and computation cost. This layer also generates a feature map of the most

Figure 3. Semantic-based deep features extraction



essential features from the previous feature set. The proposed CNN network uses softmax and dropout layers to combat overfitting. Equation 6 shows the outcome of the one-dimensional CNN network.

$$o_k^1 = f(c_k^1 + \sum_{i=1}^{N_{l-1}} \mathrm{Con1D}\left(X_{ik}^{l-1}, t_i^{l-1}\right)) \tag{6}$$

Where c_k^1 is the parameter bias of the kth neuron in the first layer, t_i^{l-1} is the result of the ith neuron in layer l-1, X_{ik}^{l-1} is the kernel strength from the ith neuron in layer l-1 to the kth neurons in layer l, and ‘‘f()” is the activation function. The CNN network is capable of extracting deep features, which aids in the development of an efficient classification model.

3.4 Multi-View Deep Learning

Usually, in the CNN model, the features extracted from pooling layers are forwarded to the fully connected layer. In the proposed approach, the deep features are forwarded to an LSTM layer, rather than a fully connected layer. The deep features are extracted from CNN, and further long-short-term correlations are detected by LSTM. To classify network traffic, the method employs a multi-view approach of the CNN-LSTM model, using the strengths of both models (Lyu & Liu, 2021; Wang, Yu, Lai, & Zhang, 2016). Figure 4 describes the combined approach of the CNN-LSTM model mainly in two steps. Step 1 involves the use of convolution and max-pooling layers; while step 2 involves the use of the LSTM layer. The CNN layers encode the LSA-based latent features, and the LSTM layers decode the information in the feature set.

LSTM relies on the concept of cell state and the numerous gates it incorporates. Knowledge about a cell’s state is transmitted beneath the sequence alignment network via the cell state. There is a main memory component in the LSTM model, as well as input, forget, and output communication gates. The memory module is vital to the designed model. The memory cell stores the previous state in its memory. The sigmoid function processes both the recent hidden state and current input. The values are 0 to 1. If the value is close to 0, it means forget gate. The input gate specifies how much network data is stored in the unit state for each time “t”. The forget gate then decides whether or not the associated records can pass to the input gate. Remember that the hidden state contains previous input data. Begin by adding the previous cell state and the received signal. The tanh function multiplies the nonlinear activation output to determine the hidden state’s relevant information. To make accurate predictions, use the hidden state’s result. 6 describes the model’s behavior.

$$i_t = \tilde{A}\left(V_{ixt} + W_i h_{t-1} + b_i\right)$$
$$f_t = \tilde{A}\left(V_{fxt} + W_f h_{t-1} + b_f\right)$$

$$c_t^{\sim} = \tanh\left(V_c X_t + W_c h_{t-1} + b_c\right) \tag{7}$$

$$c_t = \left(f_t A C_t + W_c h_{t-1} + i_t A c_t^{\sim}\right)$$
$$o_t = \tilde{A}\left(V_o x_t + W_o h_{t-1} + b_o\right)$$
$$h_t = o_t A \tanh\left(c_t\right)$$

The input at time "t" is expressed by the symbol x_t, the influence matrices are expressed by the symbols x_* and w_*, and the bias and hidden conditions are expressed by the symbols b and h. The activation functions are illustrated by the symbols σ and tanh. It is followed by the letters i_t, f_t, o_t, and c_t, which stand for input gate, forget gate, output gate, and memory cell, respectively. Collectively, these gates perform the processing of the designed model.

4. RESULTS AND DISCUSSIONS

4.1 Dataset Preparation

The proposed method is thoroughly examined using two datasets obtained from the Canadian Institute for Cybersecurity[2]. The first dataset, the Canadian Institute of Cybersecurity Android Adware and General Malware (CICAAGM2017) (Lashkari, Kadir, Gonzalez, Mbah, & Ghorbani, 2017), is collected semi-automatically through the installation of Android apps on legitimate mobile phones. A total of 1900 apps are used to create the dataset, which is divided into three categories: adware, general malware, and benign. A detailed description of each app is given in Table 1 and Table 2.

4.2 Evaluation Measures

We set widely standard training and testing ratios, such as 80%, and 20%, respectively. We used five kinds of evaluation measures such as precision, recall, f-measure, accuracy, and confusion matrix. When the model correctly predicts the benign class, this is referred to as a true positive outcome. While a true negative is an outcome in which the model correctly predicts the malware class. One type of false positive is when a researcher incorrectly concludes that a hypothesis or finding is correct when it is not (also called a type I error). If something is falsely ruled out, this is known as a false negative (also called a type II error). The general classification performance is assessed utilizing an accuracy matrix. This is equal to the sum of correctly classified instances divided by the total number of instances. The evaluation matrices are given in equations 8 and 9.

Figure 4. A multi-view deep learning approach using CNN and LSTM

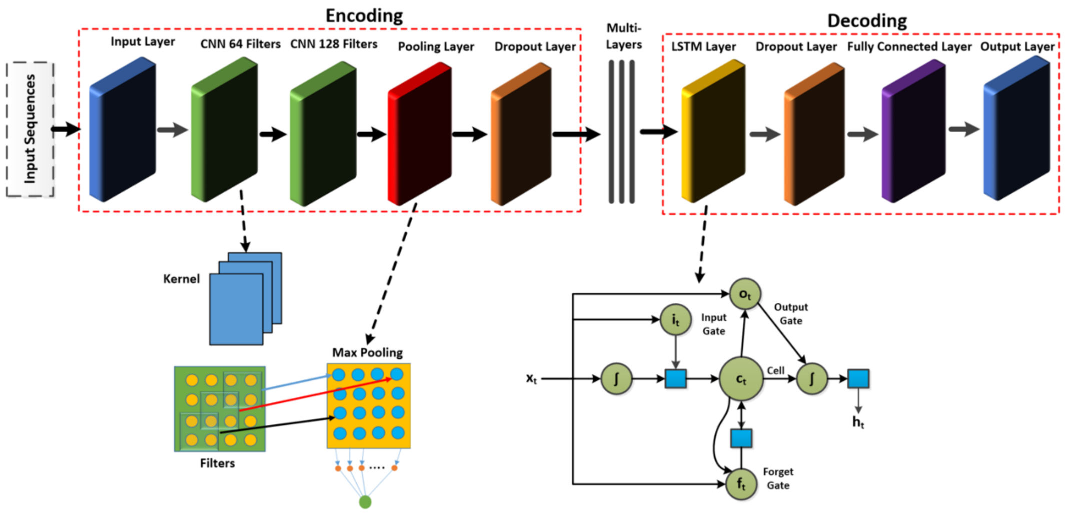


Table 1. Android Adware and General Malware Dataset (CIC-AAGM2017)

Apps	No. of Apps	Families	Description
Adware	250	Airpush	It sends intrusive ads to users' systems to steal data
		Dowgin	Works as an ad library that can also collect user data
		Kemoge	It is used to take control of the user's Android device
		Mobidash	Created to show advertisements and steal personal data
		Shuanet	It can also take control of a user's device
General Malware	150	AVpass	Distributed as a Clock app disguised as a utility
		FakeAV	Scam to trick users into buying full-version software
		FakeFlash	Built as a spoof Flash app to redirect users to a website
		GGtracker	Designed to steal information via SMS fraud
		Penetho	Fake service designed (hack tool to crack WiFi password)
Benign	1500	Benign	Clean apps (Not malicious)

Table 2. CICMalDroid 2020 dataset

Apps	Families	No. of Apps	Description
Malware	Adware	1,253	There are ads disguised inside malware-infected apps
	Banking	2,100	It is used to get access to the user's online banking accounts
	Riskware	2,546	It can be any legitimate app that can cause harm if misused
	SMS	3,904	It uses the SMS service to conduct attacks
Benign	Benign	1,795	Clean apps (Not malicious)

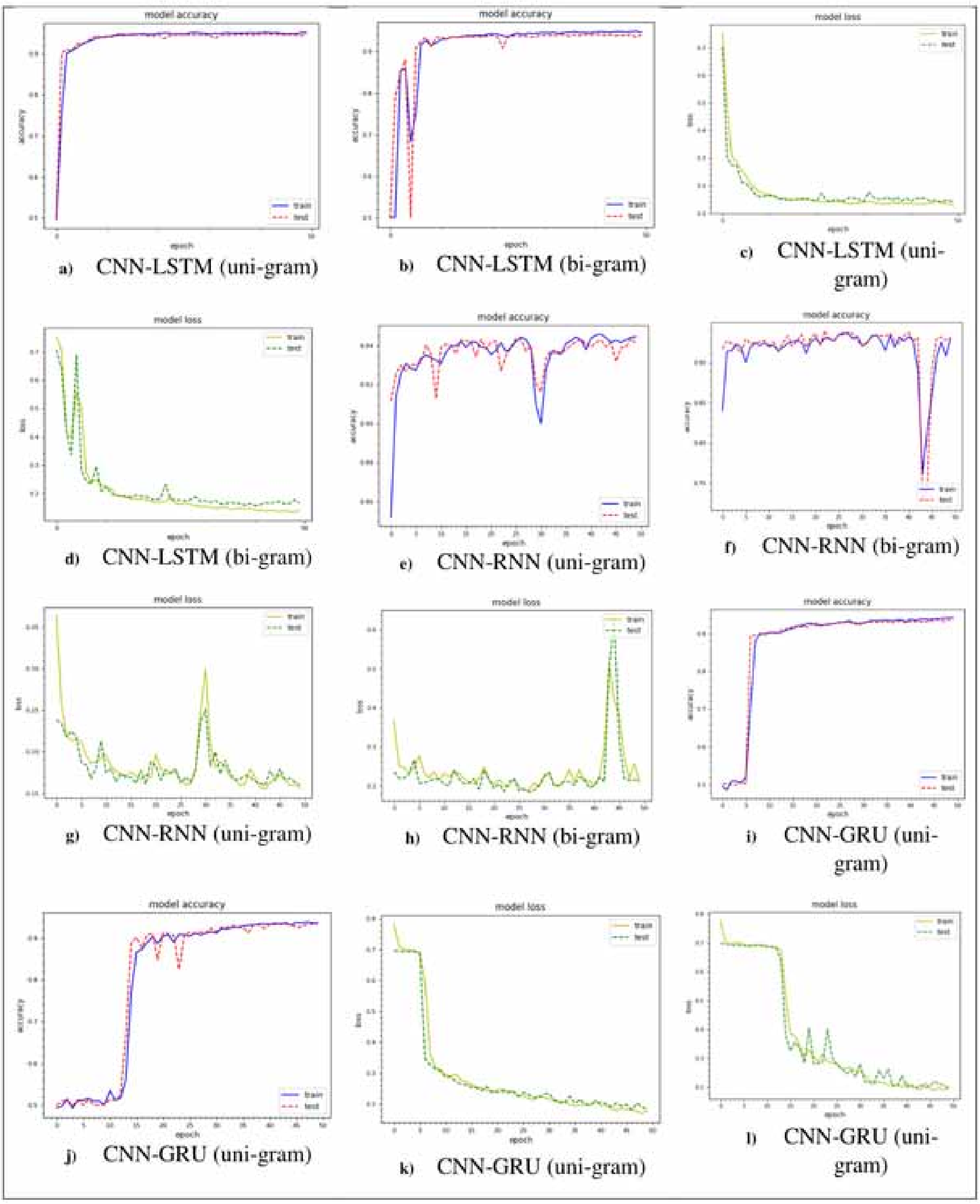
$$TPR = \frac{TP}{TP+FN}; FPR = Recall = \frac{FP}{FP+TN}, Precision = \frac{TP}{TP+FP} \tag{8}$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}, F - measure \frac{2^{*}TP}{2TP+FP+FN} \tag{9}$$

4.3 Performance Analysis

Figure 5 depicts the network-based malware classification epoch curves from the CIC-AAGM2017 dataset. To test the method, we used uni-gram, and bi-gram N-gram features to compare CNN-LSTM, CNN-RNN, and CNN-GRU models. The accuracy and loss epoch curves are used to examine overall performance. Parts (a-d) show the accuracy and loss epoch curves for the CNN-LSTM model. The CNN-RNN accuracy and loss epoch curves for uni-gram and bi-gram are shown in parts (e-h). Similarly, parts (i-l) present the accuracy and loss epoch curves for the CNN-GRU model. In part a, the training and testing curves start at 50% and gradually increase to 90%. After that, there is a sharp drop from 90% to 96.50%, followed by a more or less constant drop. Part e shows different behavior for CNN-RNN. For instance, the training and testing curves start at 40% and 91%. On the 32nd epoch, the training curve behaves at 80% while the testing curve behaves at 90%. Then it's pretty

Figure 5. Epoch curves for malware classification using CIC-AAGM2017 dataset



much constant with a 94% accuracy. A similar range of behavior is seen in Part I for CNN-GRU. The movement of these curves differs slightly when using deep features with bi-gram. For instance, CNN-LSTM, CNN-RNN, and CNN-GRU exhibit behavior ranging from 50% to 95%. More bi-gram features mean more data for training and testing. It may be overburdening the training time with noisy data. Nonetheless, the proposed approach can handle massive amounts of sequenced data by reducing and selecting features. The loss curves for uni-gram and bi-gram for CNN-LSTM, CNN-RNN, and

CNN-GRU, respectively, fall between (80%, 10%), (90%, 10%), and (40%, 15%), and (78%15%). The confusion matrices for the CIC-AAGM2017 dataset used to examine uni-gram and bi-gram for malware detection, are shown in Figure 6. The diagonal cells represent the correct classification values while the non-diagonal cells represent the incorrect classification values. As an example, in parts (a, b), CNN-LSTM has classification and miss-classification values of (91%, and 9%), respectively. Similarly, the benign has (99, 1%) and (97, 3%). Using CNN-RNN, the classification and miss-classification values for malware are (90%, 10%) and benign (98%, 2%) and (96%, 4%) respectively. Furthermore, using CNN-GRU, the classification and miss-classification values for malware is (90%, 10%), (91%, 9%) and benign are (97%, 3%) and (95%, 5%) respectively.

Figure 6. Confusion matrices for Android malware detection using CIC-AAGM2017 dataset

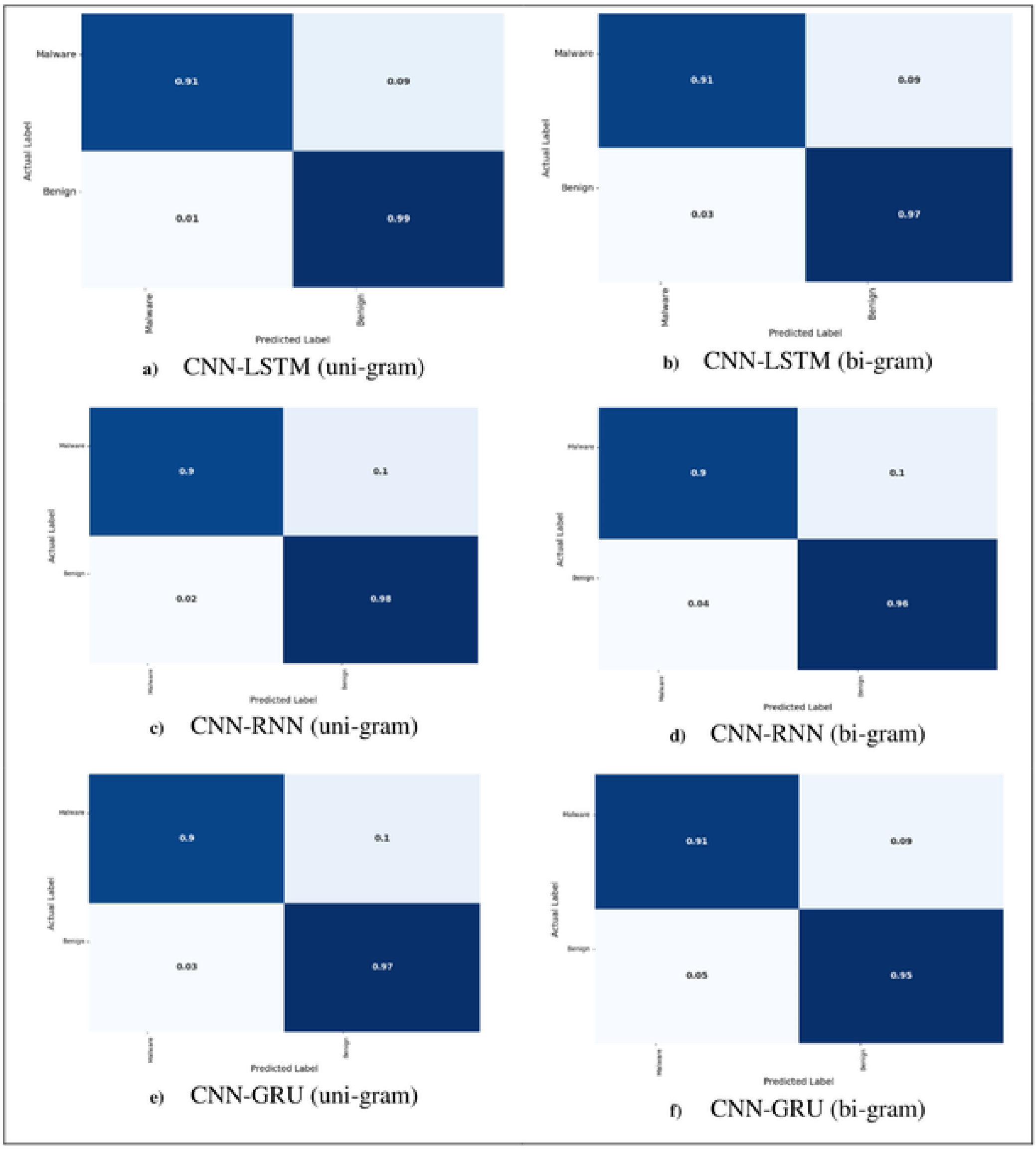
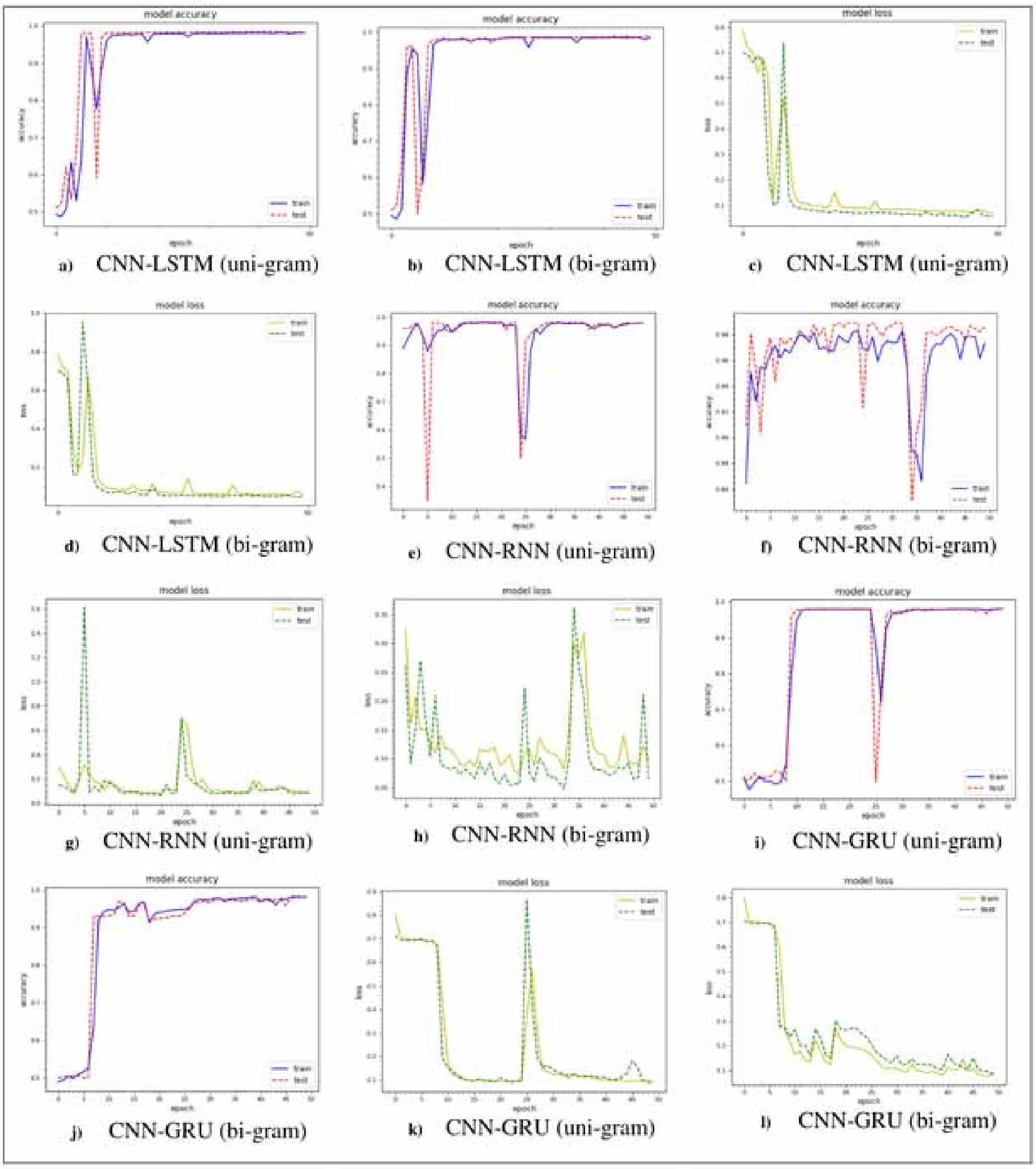


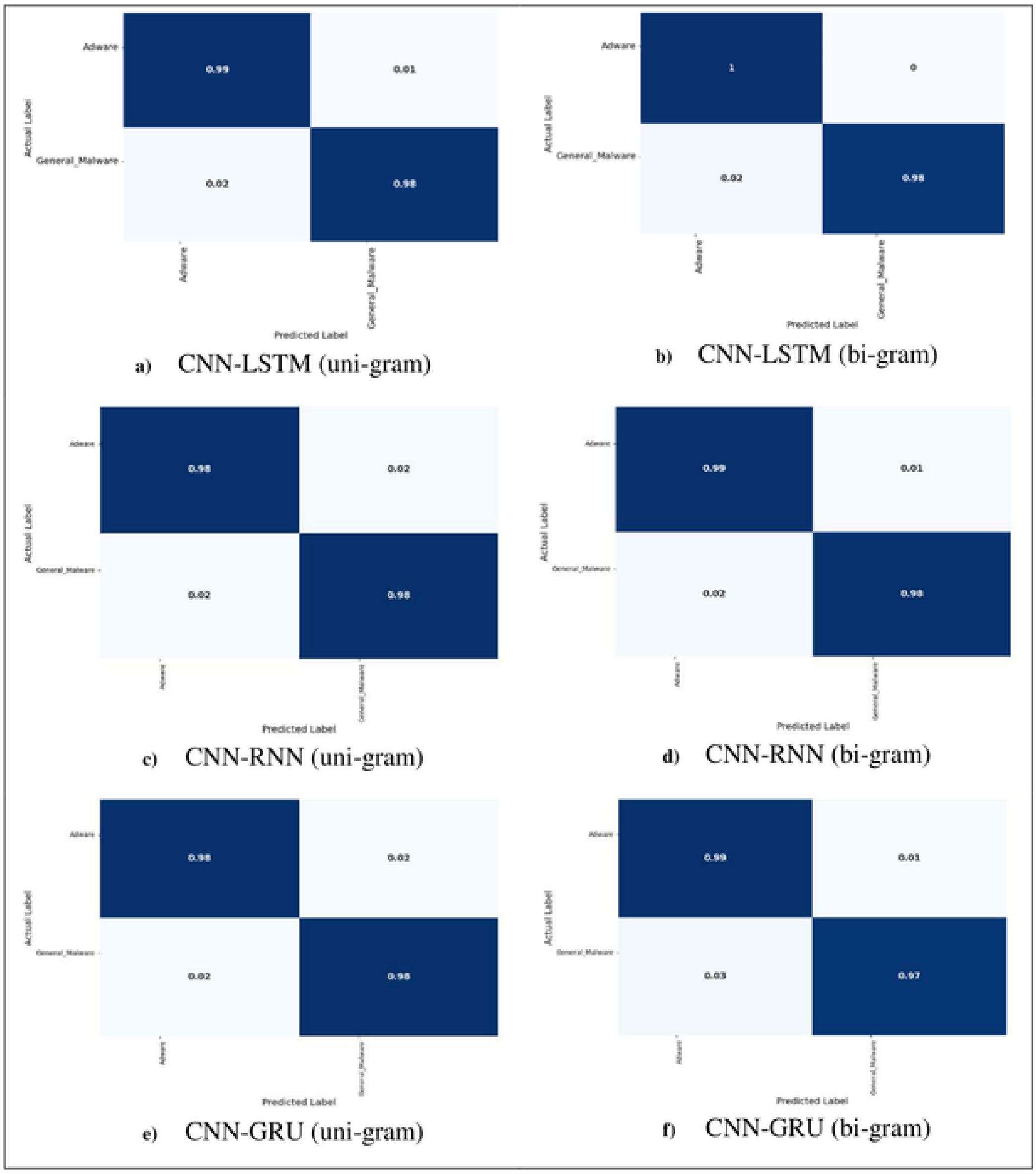
Figure 7 depicts the accuracy and loss curves for malware detection in the CIC-AAGM2017 dataset. The proposed approach detects malware more effectively than malware classification. The accuracy curves for CNN-LSTM, CNN-RNN, and CNN-GRU using uni-gram features are between (50%, 98%), (90%, 97%), and (40%, 96%), respectively. The testing curve for CNN-LSTM drops to 58% on the 10th epoch, while the training curve drops to 78%, indicating possible data loss (part c). As shown in part g, when CNN-RNN is used on the 5th and 25th epochs, the training, and test curves drop, indicating data loss. The accuracy curves for bi-gram features, CNN-LSTM, CNN-RNN, and CNN-GRU, are between (50%, 100%), (86%, and 98%). (50%, 99%), respectively. The testing and training curves in CNN-RNN parts (f, h) behave very differently, indicating data loss. These unusual

Figure 7. Epoch curves for Android malware detection using CIC-AAGM2017 dataset



behaviors can affect malware detection accuracy. On the CIC-AAGM2017 dataset, with uni-gram and bi-gram deep features analysis, confusion matrices for malware classifcation are shown in Figure 8. When CNN-LSTM is used in parts (a, b), the adware has classification and miss-classification values of (99%, 1%) and (100%, 0%) respectively. Similarly, general malware has (98%, 2%) and (98%, 2%) respectively. The classification and miss-classification values for adware are (98%, 2%) and general malware is (98%, 2%) using CNN-RNN in parts (c, d). The classification and miss-classification values for adware are (98%) (2%), (99%,1%), general malware (98%, 2%), (97%, 3%). The CNN-LSTM approach with bi-gram features has better malware detection accuracy.

Figure 8. Confusion matrices for Android malware classification using CIC-AAGM2017 dataset



The confusion matrices for malware detection using the CICMalDroid 2020 dataset are shown in Figure 9. The malware has classification and miss-classification values of (99, 1%), (96%, 4%), respectively, when CNN-LSTM is used in parts (a, b). Similarly, the benign has (99.9%, 1%), (99.9%, 1%), and (99.9%, 1%), respectively. Following that, using CNN-RNN in parts (c, d), the classification and miss-classification values for malware are (96%, 4%), (97%, 3%), and benign are (100%, 0%), (86%, 4%), respectively. Furthermore, using CNN-GRU in parts (e, f), the classification and miss-classification values for malware are (98%, 2%), (94%, 6%), and benign are (90%, 10%), (96%, 4%), respectively. The confusion matrices for malware classification using the CICMalDroid 2020 dataset with uni-gram and bi-gram deep features analysis are shown in Figure 10. Using this dataset, malware

Figure 9. Confusion matrices for Android malware detection using CICMalDroid 2020 dataset

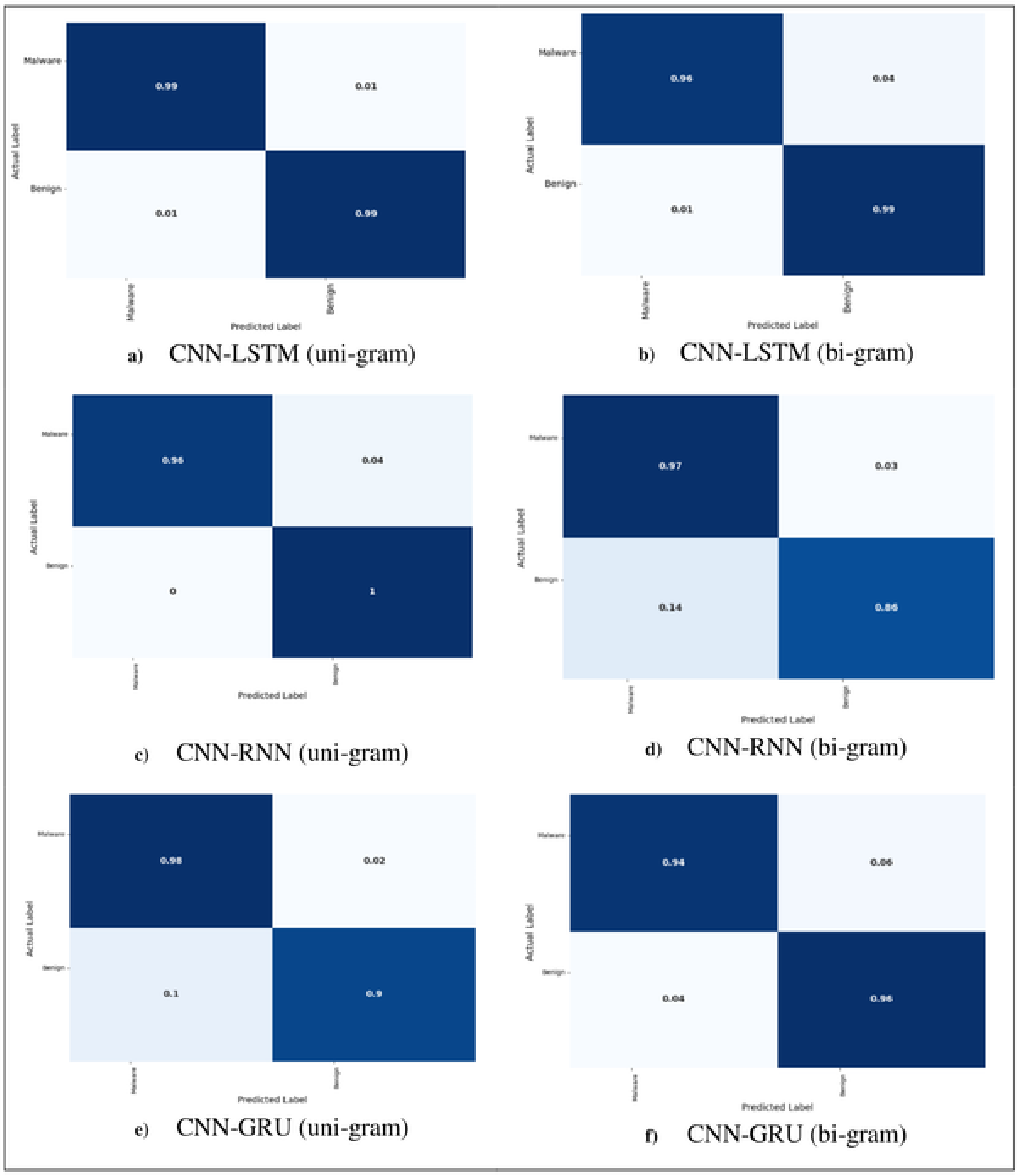
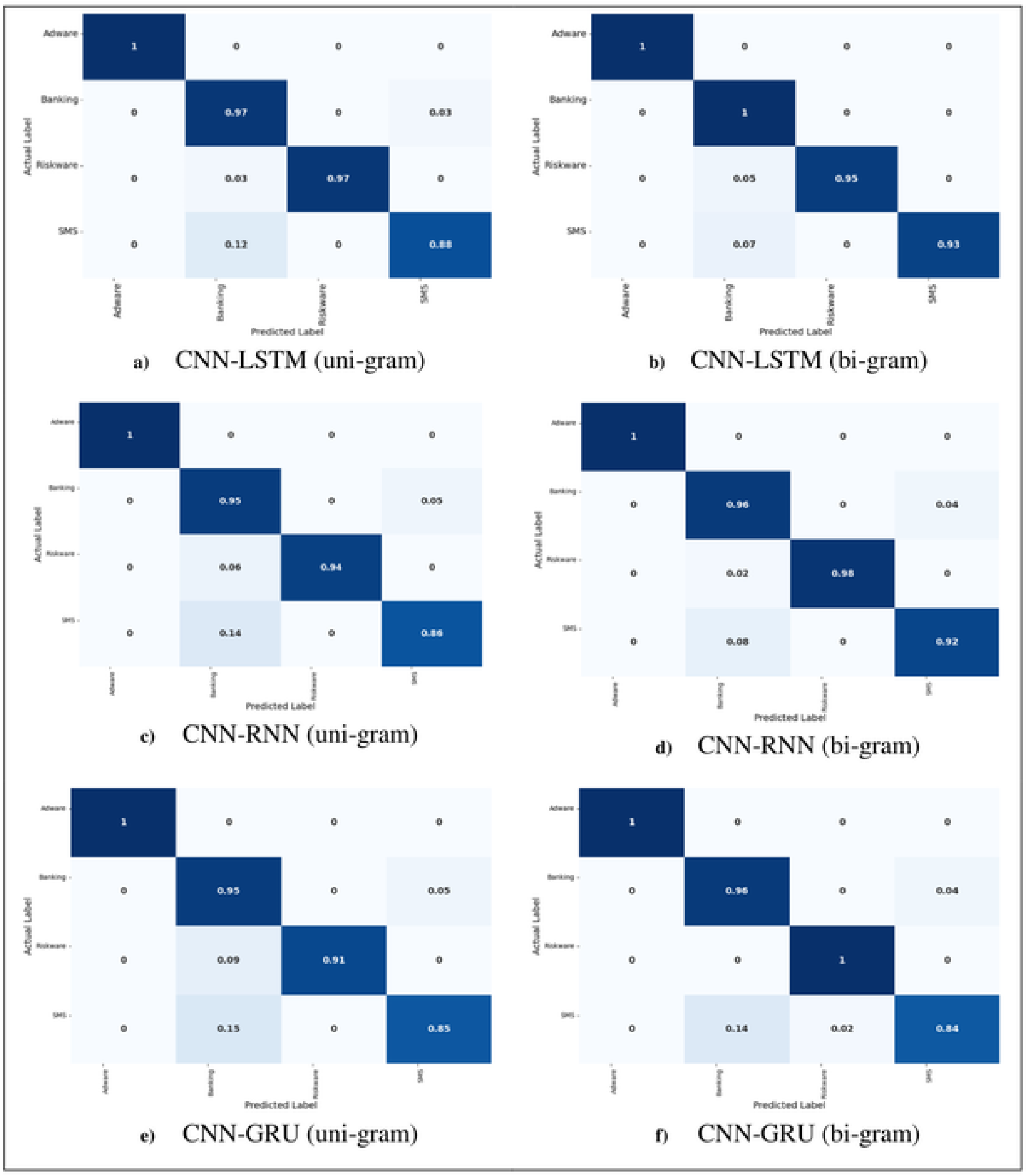


Figure 10. Confusion matrices for Android malware classification using CICMalDroid 2020 dataset



detection outperforms classification. For example, using CNN-LSTM in parts (a, b), the adware family has classification and miss-classification values of (100%, 0%), (100%, 0%), respectively. The banking family has (97%, 3%), (100%, 0%, and (100%, 0%), respectively. The riskware family has (97%, 3%), (95%, 5%), and (95%, 5%), respectively. The SMS family has (88%, 12%), (93%, 1%), and (93%, 1%), respectively. Using CNN-RNN in parts (a, b), the adware family has classification and miss-classification values of (100%, 0%), (100%, 0%), respectively. The banking family has (95%, 5%), (96%, 4%), and (96%, 4%), respectively. As can be seen, the proposed approach outperforms CIC-AAGM2017 in terms of classification results for CICMalDroid 2020.

Table 3 compares state-of-the-art malware classification methods using the CIC-AAGM2017 dataset. To analyze both uni-gram and bi-gram features, evaluation matrices such as precision, recall, f1-score, and classification accuracy are used. It can be seen that uni-gram features provide better classification accuracy for CNN-LSTM, i.e. 94.75%, than bi-gram features, i.e. 94%. Similarly, the CNN-RNN and CNN-GRU perform well with uni-gram, 94.12%, and 93.58%, respectively, as compared to bi-gram, 93.05%, and 93.27%. When compared to uni-gram analysis, bi-gram analysis generates roughly twice as many features. However, the classification accuracies of the uni-gram and bi-gram remain slightly different. It shows that our method can classify a large number of malware features with better classification results. On the other hand, malware detection results with bi-gram features outperform those with uni-gram features. For example, Table 4 compares state-of-the-art

Table 3. Comparisons for malware classification using CIC-AAGM2017 dataset

Features	Classes	Precision (%)	Recall (%)	F1-score (%)	CA (%)	Model
Uni-gram	Malware	99	91	95	94.75	CNN-LSTM
	Benign	91	99	95		
	Malware	98	90	94	94.12	CNN-RNN
	Benign	91	98	94		
	Malware	97	90	93	93.58	CNN-GRU
	Benign	91	97	93		
Bi-gram	Malware	97	91	94	94	CNN-LSTM
	Benign	92	97	94		
	Malware	96	90	93	93.05	CNN-RNN
	Benign	90	96	93		
	Malware	95	91	93	93.27	CNN-GRU
	Benign	92	95	93		

Table 4. Comparisons for malware detection using CIC-AAGM2017 dataset

Features	*Malware Families*	*Precision (%)*	*Recall (%)*	*F1-score (%)*	*CA (%)*	*Model*
Uni-gram	Adware	98	99	99	98.53	CNN-LSTM
	General Malware	99	98	99		
	Adware	98	97	98	97.82	CNN-RNN
	General Malware	98	98	97		
	Adware	98	98	98	98.15	CNN-GRU
	General Malware	98	98	98		
Bi-gram	Adware	98	100	99	99.46	CNN-LSTM
	General Malware	100	98	99		
	Adware	98	99	99	98.63	CNN-RNN
	General Malware	99	98	99		
	Adware	98	99	98	98.27	CNN-GRU
	General Malware	99	97	98		

malware detection methods using the CIC-AAGM2017 dataset. The proposed approach is analyzed using evaluation matrices such as precision, recall, f1-score, and classification accuracy.

The proposed method is tested on the CICMalDroid 2020 dataset using various evaluation matrices such as precision, recall, f1-score, and classification accuracy. Table 5 compares malware classification performance with state-of-the-art methods using uni-gram and bi-gram features. CNN-LSTM, CNN-RNN, and CNN-GRU classification accuracies with uni-gram are 98.8%, 97.93%, and 93.57%, respectively. Table 6 compares malware detection performance with the CICMalDroid 2020 dataset to other state-of-the-art methods. We have four types of malware: adware, banking malware, riskware malware, and SMS malware. The CNN-LSTM, CNN-RNN, and CNN-GRU have malware detection accuracies of 95.35%, 93.69%, and 92.73%, respectively, thanks to uni-gram deep features. The CNN-LSTM, CNN-RNN, and CNN-GRU have malware detection accuracies of 97.09%, 96.62%, and 95.1%, respectively, with bi-gram deep features. Bi-gram deep features are effective for malware detection, while uni-gram deep features are effective for malware classification.

Table 7 shows the comparisons of the proposed approach with previously published works. These works mainly used HTTP network flows to classify the Android apps as malware or benign. Multi-view neural networks are used to develop a malware analysis approach that provides depth and breadth of the features. The HTTP-based malware classification accuracy is 98.81%. Our proposed approach outperforms with a malware detection accuracy of 99.46%.

4.4 T-SNE Features Visualization for Performance Validation

The t-SNE visualization method is intended to determine whether the features contain substantial or sparse information. Moreover, the t-SNE algorithm is designed to validate the effectiveness of the proposed approach. Maaten et al. (Van der Maaten & Hinton, 2008) developed a t-SNE algorithm for visualizing high-dimensional data.

Figure 11 shows the balance of local and global semantic feature weights for various perplexity values. Two t-SNE visualization tests were created in R. We try to find the minimum level of perplexity required to distinguish malware from benign clusters. In the second test, the best malware and benign clusters are separated by perplexity. For instance, parts (a, c, e, g) show the lowest perplexity values and parts (b, d, f, e) show the optimal perplexity values. t-SNE uses iterations to distinguish between samples. To show the different malware and benign

Table 5. Comparisons for malware classification using CICMalDroid 2020 dataset

Features	Classes	Precision (%)	Recall (%)	F1-score (%)	CA (%)	Model
Uni-gram	Malware	99	99	99	98.8	CNN-LSTM
	Benign	99	99	99		
	Malware	100	96	98	97.93	CNN-RNN
	Benign	96	100	98		
	Malware	90	98	94	93.57	CNN-GRU
	Benign	97	90	93		
Bi-gram	Malware	96	98	97	97.19	CNN-LSTM
	Benign	98	96	97		
	Malware	87	97	92	91.35	CNN-RNN
	Benign	97	86	91		
	Malware	96	94	95	94.88	CNN-GRU
	Benign	94	96	95		

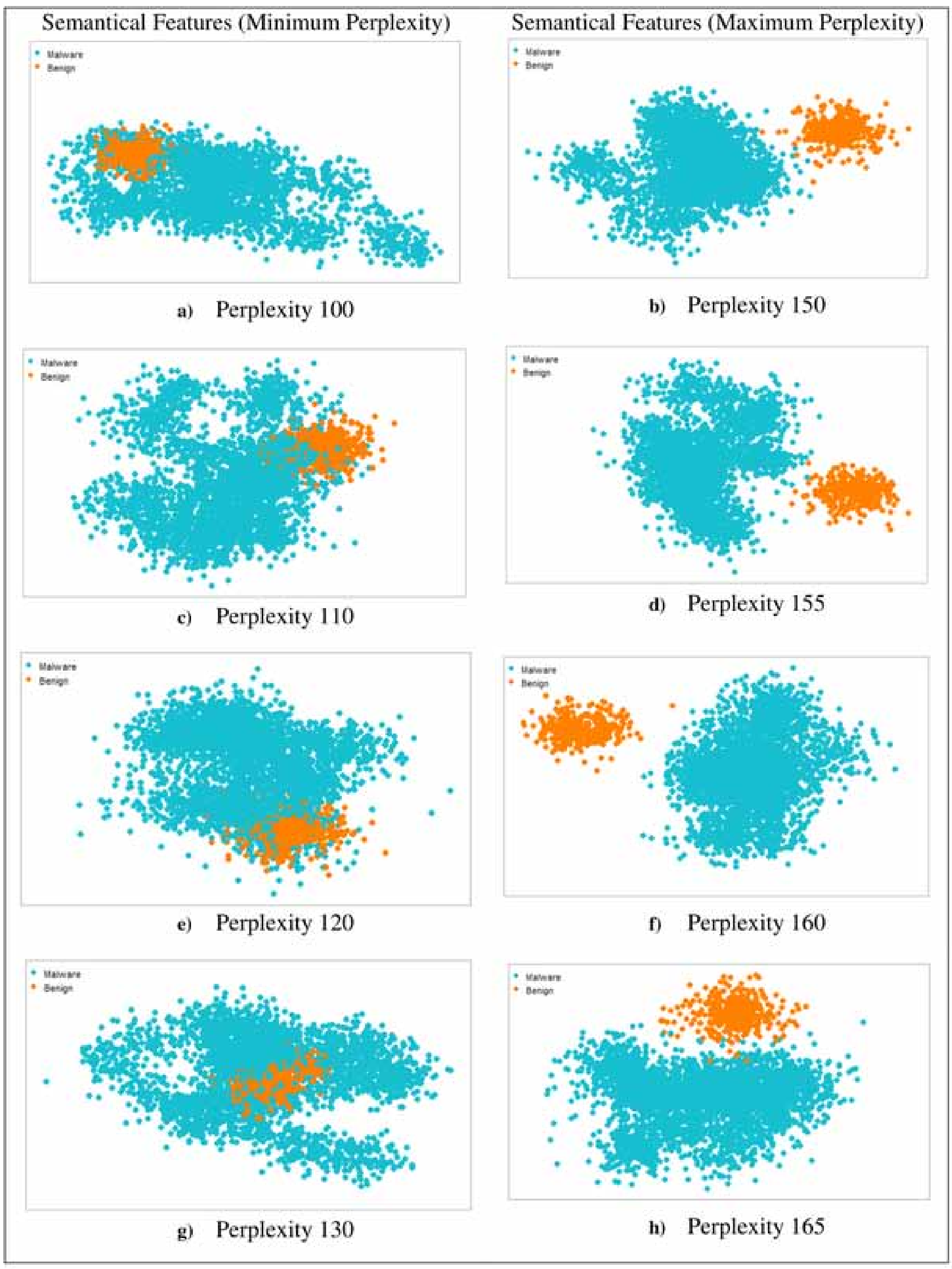
Table 6. Comparisons for malware detection using CICMalDroid 2020 dataset

Features	Malware Families	Precision (%)	Recall (%)	F1-score (%)	CA (%)	Model
Uni-gram	Adware	100	100	100	95.35	CNN-LSTM
	Banking	86	97	91		
	Riskware	100	97	98		
	SMS	96	88	92		
	Adware	100	100	100	93.69	CNN-RNN
	Banking	82	95	88		
	Riskware	100	94	97		
	SMS	95	86	90		
	Adware	100	100	100	92.73	CNN-GRU
	Banking	80	95	87		
	Riskware	100	91	95		
	SMS	95	85	90		
Bi-gram	Adware	100	100	100	97.09	CNN-LSTM
	Banking	90	100	95		
	Riskware	100	95	98		
	SMS	100	93	96		
	Adware	100	100	100	96.62	CNN-RNN
	Banking	91	96	93		
	Riskware	100	98	99		
	SMS	96	92	94		
	Adware	100	100	100	95.1	CNN-GRU
	Banking	88	96	92		
	Riskware	98	100	99		
	SMS	96	84	90		

Table 7. Comparisons with previously published works based on HTTP flows

Work	Year	Methods	Accuracy (%)
Aresu et al. (Aresu et al., 2015)	2015	Signature-based clustering	96.66
Li et al. (Z. Li, Sun, Yan, Srisa-an, & Chen, 2016)	2016	Droid Classiðer	94.66
Shanshan et al. (S. Wang et al., 2018)	2018	Skip-gram with Neural Network	95.74
Shyong et al. (Shyong et al., 2020)	2020	Random Forest	98.86
Shanshan et al. (S. Wang et al., 2020)	2020	Multi-view Neural Network	98.81
Our approach	…	N-gram with CNN-LSTM	99.46

Figure 11. t-SNE visualization for fused features using minimum (100, 110, 120, and 130) and optimal (150, 155, 160, and 165) perplexity values



clusters, we used 300 iterations for each perplexity. Semantic feature plots on 100,110,120,130 perplexity values are shown in the first column. An orange cluster indicates benign samples, while a light blue cluster indicates malware. Part a has no discernible cluster with overlapped data when perplexity is 100. On 110, 120, and 130 perplexity levels, the first t-SNE experiment shows the separation of malware and benign samples into visual clusters. As a result, the second experiment is run to distinguish malware from benign clusters. For the second t-SNE test, we chose four ideal perplexity values: 150, 155, 160, and 160. The second column shows how a low-dimensional semantic property can be used to group data into groups.

The density of the dataset influences overall classification performance. Higher density predicts better because more descriptive information is available for training. Optimal perplexity values help isolate t-SNE visual clusters for better classification. Using acceptable perplexity values, a dataset can be separated and then classified using hyperparameters. This method is used to test the proposed approach's effectiveness because semantic features can easily be separated and classified as malicious or benign.

4.5 Threat to Validity

4.5.1 Network Flow

Malware that uses protocols other than HTTP cannot be detected using the proposed method because it only looks at HTTP in the network traces. We chose HTTP for our study because it accounts for 70% of the network traffic generated by Android apps. We can simply determine the general network behavior and whether the specified activity is malicious or not by analyzing HTTP-based traffic. According to the findings of the experiments, our technique detects the vast majority of legitimate malware attacks. The proposed method can also expose suspicious network behavior of apps, which is not achievable with other methods.

4.5.2 Encrypted Traffic

Malware is increasingly choosing to communicate with networks via encrypted traffic to avoid detection. Nevertheless, the issue of encrypted malicious traffic has been examined by several studies. To deal with encrypted traffic, we devised a method based on Wireshark commands that can decrypt the encrypted network traffic into readable form. As a result, the proposed method is also capable of detecting malware samples in HTTPS traffic.

4.5.3 Sample Size

Our sample size may not be sufficient to represent the entire ecology of malware variants. However, we examined two large datasets, CIC-AAGM2017 and CICMalDroid 2020, which contain a total of 10.2k malware and 3.2K benign samples, which is significantly larger than several other related aspects. Because we are using two large standard datasets without any alteration, we can confidently state that the proposed approach can operate on other systems as well.

4.5.4 Evaluation Measures

It's even arguable that the evaluation metrics we chose aren't appropriate. To mitigate these issues, we developed prominent research metrics, such as precision, recall, f-measure, and classification accuracy. To prove the efficacy of the proposed method, we prepared a detailed evaluation that compared CNN-LSTM to CNN-RNN and CNN-GRU. Additionally, the N-gram-based LSA features and CNN-based deep features are very effective for accurate malware classification.

4.6. Theoretical and Practical Contributions

The objective of network-based strategies is to find unique features that can be used to accurately classify malicious traffic. Multiple malicious scripts may be employed by network malware to harm mobile apps. Riskware is capable of incorporating malicious bytes required for remote session hijacking. For example, network traffic may contain “application/x-javascript” coding, which the remote device will execute to prevent regular access. Similarly, adware is a type of malware that hides on the target machine and displays ads in the form of malicious scripts. Some adware monitors online activity to deliver related ads. Moreover, unresolved IP addresses may be directly connected to some malicious software. This is a common indicator of malicious intent. It is possible to identify potentially harmful scripts in terms of behavioral segmentation through the use of network traffic. It can be time-consuming and ineffective for experts to infer malware behavior from statistical features of network traffic. We use a novel feature extraction process to achieve automatic feature selection. This approach addresses the issue of feature selection while preserving high-level semantics by automatically mining URL feature representations. We used deep features analysis of the HTTP protocol because it is the most common protocol. HTTP-based encrypted data packets are collected and filtered, and afterward, the SVD method is used to extract the most notable features. The CNN-LSTM model is designed to classify network-based malware.

5. CONCLUSION

Because a large portion of Android malware uses network traffic to carry out its malicious functions, this type of malware detection strategy has the potential to be effective. A deep features analysis approach for malware classification and detection is developed in this study. The encrypted URL-based network traffic is collected and mined first before being decrypted for further analysis. Second, using segmentation and N-gram approaches, features with ordering details are extracted. The N-gram analysis greatly expands the number of features that can result in noisy data. Third, the SVD method is used to transform the features in reduced dimensional space. The LSA tool is then used to capture the semantics. These are latent features for reduced N-gram features with semantics The CNN network is used to extract deep features from latent variables to reduce computational and training costs.

The HTTP protocol analysis may not be sufficient to deal with all forms of malicious behavior. To extract data features from various types of packets, we must investigate all commonly used communication protocols. We believe that in the future, we will investigate malware network behavior using other protocols such as Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and Domain Name Server (DNS). Furthermore, we will employ transfer learning by utilizing a pre-trained model.

CONFLICT OF INTEREST

The authors of this publication declare there is no conflict of interest.

FUNDING AGENCY

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

REFERENCES

Ahmed, U., Lin, J. C.-W., & Srivastava, G. (2021). Generative Ensemble Learning for Mitigating Adversarial Malware Detection in IoT. Paper presented at the *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE. doi:10.1109/ICNP52444.2021.9651917

Aresu, M., Ariu, D., Ahmadi, M., Maiorca, D., & Giacinto, G. (2015). Clustering android malware families by http traffic. Paper presented at the *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE. doi:10.1109/MALWARE.2015.7413693

Bader, O., Lichy, A., Hajaj, C., Dubin, R., & Dvir, A. (2022). MalDIST: From Encrypted Traffic Classification to Malware Traffic Detection and Classification. Paper presented at the *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE. doi:10.1109/CCNC49033.2022.9700625

Chen, Z., Yu, B., Zhang, Y., Zhang, J., & Xu, J. (2016). Automatic mobile application traffic identification by convolutional neural networks. Paper presented at the *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE. doi:10.1109/TrustCom.2016.0077

Chiramdasu, R., Srivastava, G., Bhattacharya, S., Reddy, P. K., & Gadekallu, T. R. (2021). Malicious url detection using logistic regression. Paper presented at the *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*. IEEE.

Dai, S., Tongaonkar, A., Wang, X., Nucci, A., & Song, D. (2013). Networkprofiler: Towards automatic fingerprinting of android apps. Paper presented at the *2013 Proceedings IEEE INFOCOM*. IEEE. doi:10.1109/INFCOM.2013.6566868

David, O. E., & Netanyahu, N. S. (2015). Deepsign: Deep learning for automatic malware signature generation and classification. Paper presented at the *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE. doi:10.1109/IJCNN.2015.7280815

Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M. S., Conti, M., & Rajarajan, M. (2014). Android security: A survey of issues, malware penetration, and defenses. *IEEE Communications Surveys and Tutorials*, *17*(2), 998–1022. doi:10.1109/COMST.2014.2386139

HofmannT. (2013). Probabilistic latent semantic analysis. *arXiv:1301.6705*.

Karbab, E. B., & Debbabi, M. (2019). MalDy: Portable, data-driven malware detection using natural language processing and machine learning techniques on behavioral analysis reports. *Digital Investigation*, *28*, S77–S87. doi:10.1016/j.diin.2019.01.017

Landauer, T. K., McNamara, D. S., Dennis, S., & Kintsch, W. (2013). *Handbook of latent semantic analysis*. Psychology Press.

Lashkari, A. H., Kadir, A. F. A., Gonzalez, H., Mbah, K. F., & Ghorbani, A. A. (2017). Towards a network-based framework for android malware detection and characterization. Paper presented at the *2017 15th Annual conference on privacy, security and trust (PST)*. IEEE. doi:10.1109/PST.2017.00035

Laughlin, B., Sankaranarayanan, K., & El-Khatib, K. (2020). A service architecture using machine learning to contextualize anomaly detection. [JDM]. *Journal of Database Management*, *31*(1), 64–84. doi:10.4018/JDM.2020010104

Lee, W. Y., Saxe, J., & Harang, R. (2019). *SeqDroid: Obfuscated Android malware detection using stacked convolutional and recurrent neural networks Deep learning applications for cyber security*. Springer.

Li, W., Bao, H., Zhang, X.-Y., & Li, L. (2022). AMDetector: Detecting Large-Scale and Novel Android Malware Traffic with Meta-learning. Paper presented at the *International Conference on Computational Science*. IEEE. doi:10.1007/978-3-031-08760-8_33

Li, Z., Sun, L., Yan, Q., Srisa-an, W., & Chen, Z. (2016). Droidclassifier: Efficient adaptive mining of application-layer header for classifying android malware. Paper presented at *the International Conference on Security and Privacy in Communication Systems*. IEEE.

Lu, Y., & Da Xu, L. (2018). Internet of Things (IoT) cybersecurity research: A review of current research topics. *IEEE Internet of Things Journal*, *6*(2), 2103–2115. doi:10.1109/JIOT.2018.2869847

Lyu, S., & Liu, J. (2021). Convolutional recurrent neural networks for text classification. [JDM]. *Journal of Database Management*, *32*(4), 65–82. doi:10.4018/JDM.2021100105

McLaughlin, N., Martinez del Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., & Doupé, A. (2017). Deep android malware detection. Paper presented at the *Proceedings of the seventh ACM on conference on data and application security and privacy*. ACM. doi:10.1145/3029806.3029823

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*

Narudin, F. A., Feizollah, A., Anuar, N. B., & Gani, A. (2016). Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing*, *20*(1), 343–357. doi:10.1007/s00500-014-1511-6

Rieck, K., Holz, T., Willems, C., Düssel, P., & Laskov, P. (2008). Learning and classification of malware behavior. Paper presented at the *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. IEEE.

Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P. G., & Álvarez, G. (2013). Puma: Permission usage to detect malware in android. Paper presented *at the International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions*. Springer. doi:10.1007/978-3-642-33018-6_30

Shyong, Y.-C., Jeng, T.-H., & Chen, Y.-M. (2020). Combining static permissions and dynamic packet analysis to improve android malware detection. Paper presented at the *2020 2nd International Conference on Computer Communication and the Internet (ICCCI).* IEEE. doi:10.1109/ICCCI49374.2020.9145994

Ucci, D., Aniello, L., & Baldoni, R. (2019). Survey of machine learning techniques for malware analysis. *Computers & Security*, *81*, 123–147. doi:10.1016/j.cose.2018.11.001

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*(11).

Varma, P. R. K., Raj, K. P., & Raju, K. S. (2017). Android mobile security by detecting and classification of malware based on permissions using machine learning algorithms. Paper presented at the *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC).* IEEE. doi:10.1109/I-SMAC.2017.8058358

Vasan, D., Alazab, M., Wassan, S., Safaei, B., & Zheng, Q. (2020). Image-Based malware classification using ensemble of CNN architectures (IMCEC). *Computers & Security*, *92*, 101748. doi:10.1016/j.cose.2020.101748

Wang, J., Yu, L.-C., Lai, K. R., & Zhang, X. (2016). Dimensional sentiment analysis using a regional CNN-LSTM model. Paper presented at the *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Volume 2: Short Papers). IEEE. doi:10.18653/v1/P16-2037

Wang, S., Chen, Z., Yan, Q., Ji, K., Peng, L., Yang, B., & Conti, M. (2020). Deep and broad URL feature mining for android malware detection. *Information Sciences*, *513*, 600–613. doi:10.1016/j.ins.2019.11.008

Wang, S., Chen, Z., Yan, Q., Ji, K., Wang, L., Yang, B., & Conti, M. (2018). Deep and broad learning based detection of android malware via network traffic. Paper presented at the *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS).* IEEE, ACM. doi:10.1109/IWQoS.2018.8624143

Wang, S., Yan, Q., Chen, Z., Yang, B., Zhao, C., & Conti, M. (2017a). Detecting android malware leveraging text semantics of network flows. *IEEE Transactions on Information Forensics and Security*, *13*(5), 1096–1109. doi:10.1109/TIFS.2017.2771228

Wang, S., Yan, Q., Chen, Z., Yang, B., Zhao, C., & Conti, M. (2017b). TextDroid: Semantics-based detection of mobile malware using network flows. Paper presented at the *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS).* IEEE. doi:10.1109/INFCOMW.2017.8116346

Xu, P., Eckert, C., & Zarras, A. (2021). Falcon: malware detection and categorization with network traffic images. Paper presented at the *International Conference on Artificial Neural Networks*. IEEE. doi:10.1007/978-3-030-86362-3_10

Zhou, Y., & Jiang, X. (2012). Dissecting android malware: Characterization and evolution. Paper presented at the *2012 IEEE symposium on security and privacy*. IEEE. doi:10.1109/SP.2012.16

Zhu, H.-J., You, Z.-H., Zhu, Z.-X., Shi, W.-L., Chen, X., & Cheng, L. (2018). DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing*, *272*, 638–646. doi:10.1016/j.neucom.2017.07.030

ENDNOTES

[1] Google Play: Number of Apps 2009–2016. Accessed: May 2017. Available: https://www.statista.com/statistics/266210/

[2] https://www.unb.ca/cic/datasets/index.html

Farhan Ullah is an Associate Professor at the School of Software, Northwestern Polytechnical University, Xi'an Shaanxi, P.R. China. He received Ph.D. Computer Science degree in 2020 from College of Computer Science, Sichuan University Chengdu, P.R. China. He was awarded a full-time Chinese Government Scholarship (CGS) for his Ph.D. During his Ph.D. studies, he participated in different research projects which include, the National Key Research and Development Program, the National Natural Science Foundation of China, and the Technology Research and Development Program of Sichuan, China. He performed as a research member of the science and technology project of Taicang (2020), Jiangsu, China. He is the Special Issue Lead Guest Editor for Security and Communication Networks Journal in the CCF C category. He also served as a Guest Editor (GE) for a special issue of Future Internet Journal, MDPI. He performed as a Program Chair (PC) and Technical Committee member at an international conference on Future Networks and Distributed Systems (ICFNDS-2019), the University of Paris, France. He received Research Productivity Award (RPA) from COMSATS Institute of Information Technology (CIIT), Sahiwal, Pakistan in 2016. His research work is published in various renowned journals of IEEE, Springer, Elsevier, Wiley, MDPI, and Hindawi.

Xiaochun Cheng won a full scholarship for his university education. He received the BEng degree in Computer Engineering in 1992, Ph.D. in Computer Science in 1996. He visited the Queen's University of Belfast between 1997 and 1998. He was a Postdoc Research Associate at Sheffield University between 1998 and 2000. He was a Lecturer at Reading University between 2000 and 2005. He has been a Senior Lecturer since 2006 and since 2012 the Computer Science Project Coordinator in Middlesex University. One project was funded with a 16 Million Euro budget. He is a member of IEEE SMC Technical Committee on Intelligent Internet Systems, IEEE Communications Society Communications and Information Security Technical Committee, IEEE Technical Committee on Cloud Computing. He won the National Science and Technology Advance Award.

Leonard Mostarda is an Associate Professor and former HoD at the Computer Science department at Camerino University, Italy. He got his Ph.D. in 2006 at the Computer Science Department of the University of L'Aquila. Afterward, he cooperated with the European Space Agency (ESA) on the CUSPIS FP6 project to design and implement novel security protocols and secure geotags for works of art authentication. To this end, he was combining traditional security mechanisms and satellite data. In 2007 he was Research Associate at the Computing Department, Distributed System and Policy Group, Imperial College London. There he was working on the UBIVAL EPRC project in cooperation with Cambridge, Oxford, Birmingham, and UCL for building a novel middleware to support the programming of body sensor networks. In 2010 he was Senior Lecturer at Middlesex University in the Distributed Systems and Networking Department. He funded the SSI LAB that performs basic research in various aspects of the design, implementation, analysis, and evaluation of distributed systems. The lab conducts research with systems at all scales, from sensors devices to cloud computing data centers, and takes an experimental system approach in building real systems to investigate new research ideas. I am also CEO of the bilancioCO2zero spinoff. An innovative company for building energy-efficient solutions and reducing CO2 emissions. Particular areas of interest include wireless sensor networks, networking, middleware, and smart environment programming.

Sohail Jabbar is an Associate Professor at the Department of computational sciences, The University of Faisalabad, Faisalabad, Pakistan. He was an Assistant Professor with the Department of Computer Science and the Director of Graduate Programs at the Faculty of Sciences, National Textile University, Faisalabad, Pakistan. He was also a Postdoctoral Researcher with Kyungpook National University, Daegu, South Korea. Moreover, he was a Postdoctoral Researcher with the Department of Computing and Mathematics, Manchester Metropolitan University, Manchester, U.K. He is also the Head of the Network Communication and Media Analytics Research Group, National Textile University. He also served as an Assistant Professor with the Department of Computer Science, COMSATS Institute of Information Technology (CIIT), Sahiwal, and also headed the Networks and Communication Research Group at CIIT. He has authored one book, two book chapters, and more than 70 research papers. His research work is published in various renowned journals and magazines of the IEEE, Springer, Elsevier, MDPI, Old City Publication, Hindawi, and the IEEE conference proceedings ACM, and IAENG. He is on collaborative research with renowned research centers and institutes worldwide on various issues in the domains of the Internet of Things, wireless sensor networks, and big data. He has been a reviewer for leading journals, including the ACM TOSN, JoS, MTAP, AHSWN, and ATECS, and conferences, including the C-CODE 2017, ACM SAC 2016, and ICACT 2016. He is also a TPC member/chair for many conferences. He received many awards and honors from the Higher Education Commission of Pakistan, Bahria University, CIIT, and the Korean Government. Among those awards, the Best Student Research Awards of the Year, the Research Productivity Award, and the BK-21 Plus Post Doctoral Fellowship are few. He received the Research Productivity Award from CIIT, in 2014 and 2015, respectively. He has been engaged in many National and International Level Projects. He is also the Guest Editor of the Sis in Concurrency and Computation Practice and Experience (Wiley), the Future Generation Computer Systems (Elsevier), the Peer-to-Peer Networking and Applications (Springer), the Journal of Information and Processing System (KIPS), the Cyber-Physical System (Taylor & Francis), and the IEEE WIRELESS COMMUNICATIONS (IEEE Communication Society).