IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Revue d'Intelligence Artificielle
Vol. 36, No. 5, October, 2022, pp. 761-767
Journal homepage: http://iieta.org/journals/ria

Android Malware Classification Using LSTM Model

Nagababu Pachhala[1*], Subbaiyan Jothilakshmi[1], Bhanu Prakash Battula[2]

[1] Department of Information Technology, Annamalai University, Annamalainagar 608002, Tamil Nadu, India
[2] Department of CSE, KKR & KSR Institute of Technology & Sciences, Guntur 522017, Andhra Pradesh, India

Corresponding Author Email: nagababupachhala2024@gmail.com

https://doi.org/10.18280/ria.360514

Received: 13 July 2022
Accepted: 6 December 2022

Keywords:
malware, Android, deep learning, LSTM

ABSTRACT

From last two decades, smartphone use is essentially widespread around the world, and Android is the most popular open-source operating system, with the largest market share and active user population of any open-source operating system. This has resulted in malicious actors turning their attention toward the Android operating system to exploit user reliance and vulnerabilities that exist inside the system. Hackers can take advantage of consumers' sensitive data to engage in advertising, extortion, and theft. Most of the existing anti-malware software's cannot be able to detect all the malwares because of the intelligent malwares. In this paper we use the deep learning based Long short-term memory (LSTM) network for android malware classification. The model is effective in classification of intelligent malwares. The proposed model is implemented using google colab. The model is archiving more than the 98% accuracy in classification of android malwares.

1. INTRODUCTION

Today, cyber assaults [1] have risen to the top of the most crucial problems contemporary civilization is dealing with. According to ISO/IEC 27000:2009, these attacks are an "attempt to damage, expose, modify, disable, steal, or obtain unauthorized access to or make unauthorized use of anything that has value to the organization." The efforts devoted to combating these assaults have incurred significant expenditures, which totaled $86.4 billion in 2017 due to the activities. As diverse as these cyber-attacks [2] are presented and conducted, many processes and tactics may be used to counter and defeat them. These assaults have a lengthy history, dating back to the 1970s with the introduction of the first viruses. They have taken on many forms throughout the years, adopting complicated processes, attempting to bypass antivirus and reach the victim, or employing clever obfuscation techniques intended to keep them from being detected. The scope and complexity of the current malware issue represent a significant challenge. Malware has shown to be a potent weapon for launching wide-scale assaults against many people and systems simultaneously.

Recently, ransomware assault [3], which infected thousands of machines that were not correctly updated, encrypted data, and demanded payment in bitcoin, is an example of hackers made malware attacks While dealing with this kind of assault, there are two main tasks to be completed: the development of malware detectors capable of filtering and classifying suspicious samples that include harmful bits of code; and the mitigation of damages resulting from malware that has managed to evade the detection system. This paper is concerned with the former issue, which is addressed as an offline job in which suspicious samples are examined to determine if they are harmful or innocuous. Aside from that, malware may be found not only on personal computers but also on virtually any smart device that we use daily. Therefore, when considering the vast number of gadgets in our environment and their critical role in our everyday lives, the issues connected with malware become more acute. Particularly, our personal and sensitive information has been compromised via mobile gadgets. Smartphones [4] are the most prominent example of these gadgets. We save pictures, texts, and many other personal data, banking, medical, and other apps in mobiles. The security [5] of these devices from malware is a time-consuming and challenging job. When it comes to these devices, Android is both the mobile OS with the shares globally and the mobile web browser, accounting for close to 80% of all mobile operating systems. And the most targeted platform for malware development, accounting for 99 percent of all mobile malware.

As a result, Android malware has taken on various forms, such as scareware and ransomware [6], and has given several distinct malware families. It is critical for Android malware analysis and understanding to categorize and group dangerous apps into sets that share similar behavioural patterns and intents. Base Bridge, Plankton, Just, and Fake Run are just a few of the malware families that have resulted in hundreds of distinct programs that have successfully infected devices all over the globe, even though the variations between them are minor. Currently, Android malware is a severe, massive, and difficult-to-solve issue. The design and implementation of filters that correctly determine whether a suspicious sample is benign or dangerous are the primary focus of efforts devoted to malware countermeasures. It is feasible to assess the range of activities that the application can do and make a judgment based on a series of behavioral indicators that have been extracted. The significance of this assignment cannot be overstated. The enormous number of new applications discovered every day, on the other hand, makes the use of tools capable of dealing with vast quantities of samples on an automated basis a need. As a result, it is essential to research systems that can automate this process, thus preventing

malware samples from reaching users. As a result of this situation, machine learning methods emerge as a vital tool for addressing the issue. They may install malware detectors that can handle large applications and differentiate between harmful and benign software based on a prior training procedure using previously labeled samples. This article uses the Long short-term memory model to classify malware in android systems. Section 2 summarizes the existing literature, section 3 describes the proposed model, section 4 deals with the experimental results and section 5 concludes the paper.

2. RELATED WORK

At this point, the subject of Android malware detection has garnered a great deal of interest in the scientific literature. But only a few studies have concentrated on the machine learning methods utilized. According to our experience, none of them has developed a precise description of mobile malware detection systems based on the metrics and machine learning methods used thus far. In this part, which focuses on 2017 through 2021, such literary efforts are chronologically identified and then placed concerning the present state of art.

In their comprehensive study of the detection of malware dynamic mobile methods, Yan et al. [7] summarised various basis and performance assessment metrics for the detection of mobile malware and presented their findings. In addition, the authors conducted a study and comparison of the previously available Mobile malware detection system on the analytical techniques and the assessment results they developed. Finally, the authors identified unresolved problems in the area and potential study paths for the future.

Odusami et al. [8] attempted to uncover loopholes and offer insight into viable countermeasures against unknown Android malware using mobile malware detection methods. Their research discovered that practices that depend on machine learning to identify harmful applications were more trusted and generated better detection accuracy when compared to sign-based approaches.

Kouliaridis [9] conducted a comprehensive assessment of the literature on MMDS and classified each piece of research according to a specific categorization system. To be more precise, the latter categorizes each piece of work according to its target platform, feature selection approach, and detection methods, either signature-based or anomaly-based.

A thorough review of Android malware detection methods that use machine learning techniques was provided by Liu et al. [10]. The authors conducted in-depth analyses and presented summaries of several essential issues, including sample collection, data pre-processing, feature selection, machine learning models, algorithms, and detection performance. Finally, they discussed the limitations of machine learning methods and provided insights into possible future paths.

Gibert et al. [11] reviewed well-known machine learning methods for the detection of malware, focusing on deep learning techniques. The authors discussed the research difficulties, limitations of legacy machine learning methods, current trends, and advances in the area, emphasizing deep learning schemes.

Shabtai et al. [12] developed a system to identify malicious activity by analyzing network data. It is accomplished by recording user-specific network traffic patterns for each evaluated program and detecting variations from these patterns that may be reported as potentially harmful. They used the C4.5 method to assess their model, and they achieved an accuracy of up to 94 percent.

To identify Android malware, Canfora et al. [13] proposed an algorithm that analyses opcode normal histograms; this is done by monitoring the frequency with which each grouping of opcodes appears on the Android operating system. To be more specific, their detection algorithm uses a vector of characteristics derived from eight Dalvik opcodes to identify targets. These opcodes are often utilized to change the control flow of an application.

Jang et al. [14] developed Andro-AutoPsy, an antivirus system that detects and removes malware using Android malware similarity matching. The authors obtained malware information from antivirus mobile threat alerts, malware repositories, and community websites by searching for and collecting malware information from these sources to train the suggested model. The researchers chose five distinct footprints for the characteristics: digital certificate's serial number, API call sequence, permissions required to initiate the call, the intents, and system instructions. Andro-Autopsy, according to the authors, can detect zero-day malware. Tests were conducted on Andro-Autopsy against approximately 1K malware applications obtained from the VirusShare [15] and Contagio [16], as well as against more than 109K benign samples gathered from Google Play [17] and the Android Market [18] and the VirusShare [15] mobile datasets.

Yerima [18] created a community classification model that takes the use of critical Android and Java API calls extracted from the source code and application permissions collected from the manifest file to create a community classification model. All of the tests were conducted using McAfee's internal (private) dataset. There were several different classifiers employed during the evaluation phase.

Coronado [19] created an algorithm-based method for detecting mobile malware. Static analysis was used to detect the malicious applications' privileges and objectives using a corpus of 1531 malware programs from the Drebin dataset [20] and 765 innocuous apps. The Random Forest and Random Committee algorithms were used, with the former reaching up to 97.5 percent accuracy and the latter getting up to 97.5 percent.

To detect malignant patterns, Milosevic and colleagues [21] developed an extraction method that was focused on obtaining non-trivial and beneficial ways that might be utilized to identify malignant patterns. The M0Droid corpus [22] was considered during the experiments. C4.5, Random Forest, Naive Bayes, Support Vector Machine (SVM), JRip, and Logistic Regression were the classifiers employed in the evaluation procedure [23]. Their results produce 10% better compared to other models.

A framework for improving deep neural networks against adversarial malware was proposed by Li et al. [24]. Authors propose a defensive design that comprises many components to improve the correctness of deep neural networks against malware attacks.

According to Athiwaratkun et al. [25], iterative neural network designs were used to capture better long-term associations in API call traces to increase performance. They tested it with language models such as the Long Short-Term Memory (LSTM) and the Gated Repetitive Unit (GRU). The authors recommend a two-step procedure. The first step is to build the features associated with a certain API call tracing using the LSTM or GRU. In the second stage, these features

are classified using Logistic Regression with a single fully connected layer or SoftMax. The authors also suggested replacing the present convolutional neural network with a character-level convolutional neural network.

Gopi and Naik [26] proposed a Hierarchical Convolutional Neural Network (HCNN) [27] to cope with the hierarchical structure of portable executables, calling it a "more efficient solution." Instead of viewing malware as a stream of instructions, their research grouped the instructions together in the same function to retain the hierarchical structure of a computer program. As a result, assembly language instructions were divided into operations, each of which was represented by a collection of mnemonics, which were then concatenated to make a single set of instructions. As a result, the hierarchical CNN [28] gathered features from the data at both the mnemonic and functional levels.

3. LONG SHORT-TERM NETWORK

The LSTM was selected as the foundation model for this study. It is excellent at storing more extended information periods and is demonstrably more correct than other models which are based on sequences that are previously evaluated in the literature. The LSTM had also been used in NLP, namely for parsing of sentences and document categorization. A similarity may be drawn between the hierarchy of words to sentences and paragraphs to texts and the order of fundamental blocks to short, middle, and long-term sequences in programming. When remembering word probabilities beyond the phrase level, the normal RNN has limited capacity. Still, by recalling bursts of short-term sequential information over a more extended period, the LSTM significantly improves its power compared to the regular RNN. This improves the LSTM's ability to anticipate the words that will follow in phrases and paragraphs, but not in whole texts. The method of dealing with malware data is described in Figure 1.

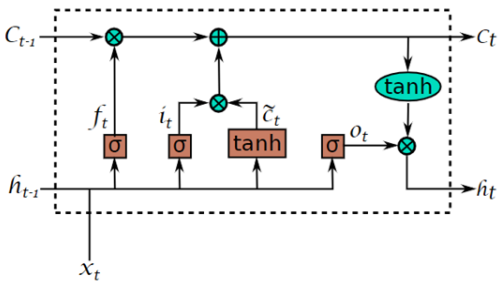

Figure 1. Cell structure in LSTM

The cell figure of the LSTM model is depicted in Figure 2. First and foremost, we will determine what information we will discard from the cell's current state. The "forget gate layer," also known as the sigmoid layer (gu), is used to make this decision. It examines the values of "i_{u-1}" and "y_u" and produces a number between 0 and 1 for every number in cell state D_{u-1}, as shown in the following example. A 1 indicates that something should be kept fully, whereas 0 means that something should be removed entirely. With weight (x_h) applied to information [i_{u-1}, y_u] and a y-intercept (c_g), as indicated in Eq. (1), the forget gate layer matches the sigmoid layer() of input in the example that attempts to guess the succeed word based on all before ones, as shown in Eq. (1). A cell state that includes the gender of the current subject may be included in such a problem for the proper pronouns to be utilized. When we come into a new trial, we would want to forget about the gender of the previous subject in question.

$$g_u = \sigma\,(X_h.\,[i_{u-1}, y_u] + c_g) \tag{1}$$

The next step is to find what new data will be stored in the present state of the cell. This has two components. The gate layer(i_t) is the first sigmoid layer and is responsible for finding values to update. The tenth layer then creates a vector of new values (t) added to the state of the system. In the next step, we will combine these two to create a new status version. As shown in Eq. (2), the gate layer (i_t) is the output of the sigmoid layer (σ); where the input is the weight (w_i) applied to the [h_{t-1}, x_t] and the y-intercept (b_i). Also, the new candidate values $\hat{C}_t$ are the output of the tanh layer (tanh); where input is the weight (W_C) applied to the input [h_{t-1}, x_t] and the y-intercept (b_c). To use the language model as an example, we want to replace the old subject with the new one and add the new subject's gender to the cell state. A transition from the previous cell state C_{t-1} to the new cell state C_t is now required. The previous stages determine our actions, and we must follow through.

$$j_u = \sigma(X_j.\,[\,i_{u-1}, y_u] + c_j) \tag{2}$$

$$D_u = \tanh\,(X_D.\,[i_{u-1}, y_u] + c_D) \tag{3}$$

Multiply the previous state by g_u, while keeping in mind the items we chose to ignore before the procedure. Then we add $j_u * D_u$. We would drop the old subject's gender information in the language model and add the new information. As shown in Eq. (4), the new cell state (d_u) is based upon the sum of forgetting gate output from the last cell state (d_{u-1}) and input gate output from the new candidate vector (D_u).

$$D_u = g_u * D_{u-1} + j_u * D_u \tag{4}$$

The final step is determining the output to be produced. The output will be based on the current state of our cell, but this will be a simplified version of the state. We apply a sigmoid layer that determines which parts of the cell state we will extract and which parts we will not. After that, we pass the cell state through tanh and multiply it by the output of the sigmoid gate to output only the designated parts. As shown in Eq. (5), the output (p_u) is the output of the sigmoid layer; where input is the weight (x_p) applied to the input [i_{u-1}, y_u] and the y-intercept (c_p). As shown in Equation. As shown in Figure 6, this final output is based on the product of the sigmoid layer output (p_u) and the tanh layer output.

$$P_u = \sigma(X_p.[i_{u-1}, y_u] + c_p) \tag{5}$$

$$i_u = P_u * \tanh\,(D_u) \tag{6}$$

4. PROPOSED WORK

In this study, we propose to employ deep neural networks for malware classification and analysis via the use of API calls. The major focus of this research is on the LSTM network structure, which extracts the properties of Android APKs and creates a classification that can be used to identify possible

Android malware without the requirement for the application source code to be utilized. Here Figure 1 explains the proposed architecture of the android malware detection model. Initially, it takes the input data from Durbin data and assigns it to the neurons as weights, it trains with the malware as well as non-malware samples. It checks if it is the end of the input sequence stop training and ends the training or else continues till the end of the dataset. The training is done by comparing malware and non-malware conditions if it is expected one gets trained or else it leaves.

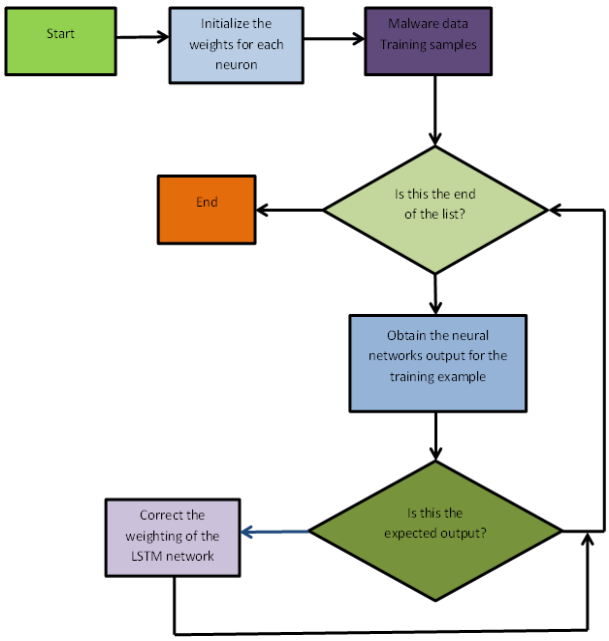

Figure 2. Flow chart of the proposed model

Algorithm LSTM Malware detection ()

1. Initialization
    a. ct = [0, 0, 0]
    b. ht = [0, 0, 0]
2. for input in Dataset:
3. ct, ht = LSTM Malware (ct, ht, input)
4. Def LSTM Malware (prev_ct, prev_ht, input)
    a. Combine = prev_ht + input
    b. ft = forget_layer (combine)
    c. candiate = candiate_layer (combine)
    d. it = input_layer (combine)
    e. ct = prev_ct * ft +candiate * it
    f. ot = output_layer (combine)
    g. ht = ot * tanh(ct)
    h. return ht, ct

First, the prior concealed state and the current input are concatenated to create the current input. We'll refer to it as a combination. The combined layer is given the information from the forget layer. This layer eliminates information that isn't relevant. The combined command is used to build a candidate layer. The candidate contains a list of potential values that might be added to the cell state. The result of the combination is also supplied into the input layer. This layer determines whatever data from the candidate should be included in the new cell state. It is a decision layer. The cell state is determined utilizing the vectors obtained from the forget layer, the candidate layer, and the input layer, as well as the previous cell state once they have been computed. The last step is to calculate the output. Seventh, we may find the new hidden state by multiplying the output by the new cell state.

5. EXPERIMENTAL RESULTS

We experiment with the model using 8GB RAM windows-10 OS and Google colab interface, and python-3 as a programming language with Keras and TensorFlow packages. And Drebin-215 dataset is used for experimenting with existing and proposed models.

A. Dataset:

Drebin-215 dataset is used for experimenting with existing and proposed models. The dataset includes 215 features from 15,036 app samples, with 9476 of them being benign and 5560 being malware samples from the Drebin project [4].

B. Results and discussion

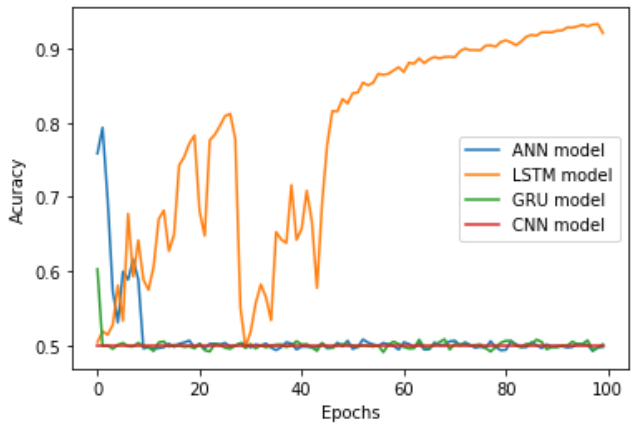

Figure 3. Accuracy

$$Accuracy = \frac{(tp + tn)}{(tp + tn + fp + fn)}$$

Here Figure 3 represents the accuracy of the classification of malevolent and benevolent samples using proposed LSTM and existing models. And the graph compares the existing ANN model and the proposed LSTM model. ANN model fails to perform classification of harmful and harmless code. Because malicious code is a series of actions performed and ANN fails to memorize the code sequences. But the LSTM model contains a memory unit that can give improved accuracy when the quantity of epochs rises. At the same time, ANN fails to provide enhanced accuracy while the number of ages is enlarged. CNN is not apt for text data processing, producing less than 50% accuracy. GRU facing gradient descent problem delivers less accuracy.

Figure 4. Precession

The greater the number of FPs introduced into the mix, the uglier that precision will seem.

Precision = TP / (TP + FP)

Here Figure 4 represents the classification of malicious samples and benign samples precession. And the graph compares the existing ANN, CNN, GRU models and proposed LSTM model. The precession of ANN model is between 0.4 and 0.6 because malicious code is a series of actions, ANN fails to memorize code sequences. But the LSTM model contains a memory unit it can give better precession while the number of epochs increases. CNN is not apt for text data processing, producing precession less 0.2. GRU facing gradient descendant problem produces less precession.

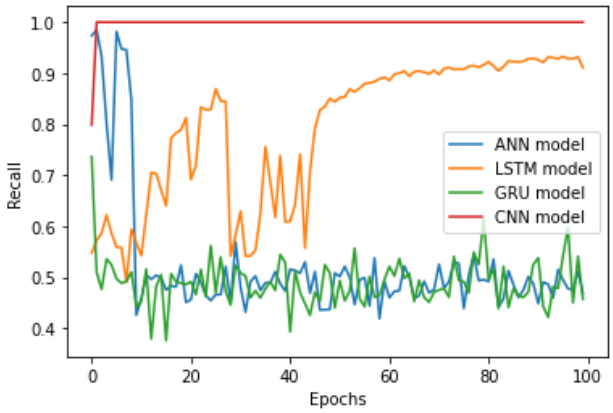

Figure 5. Recall

The recall is calculated as the quantity of accurate optimistic findings separated by the total amount of appropriate samples.

Recall = TP / (TP + FN)

Here Figure 5 represents recall for the classification of malicious data. On X-axis of the graph represented the number of epochs and Y-axis shows the recall. Here the graph shows the existing CNN, ANN GRU models, and the proposed LSTM model. GRU model suffering from the gradient exponent problem it is underperformed, ANN and CNN give recall between 0.5 to 0.9. due to the limitation of handling text data these two models are not performing better compared to the proposed LSTM model.

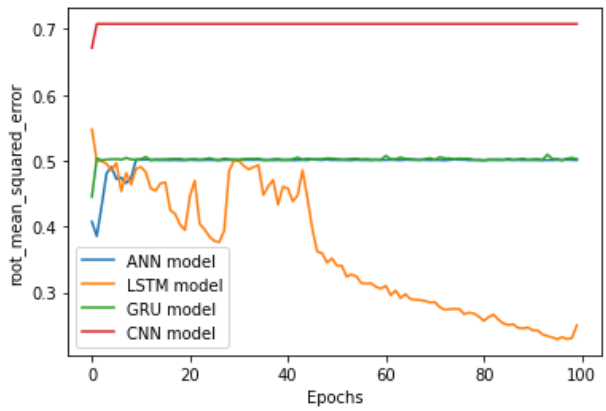

Figure 6. RMSE

In statistics, the standard deviation of the mistakes that occur when a prediction is made on a dataset is the RMSE.

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(y_j - \check{y}_j)^2}$$

This is the same as MSE, except that the basis of the number is considered when evaluating the model's accuracy. Here Figure 6 represents the RMSE of proposed LSTM and existing CNN, ANN, GRU models. Existing models ANN, RMSE is consistent between 0.3 to 0.5 while increasing the number of epochs because ANN is not good in handling text data. GRU error value between 0.4 to 0.6 and CNN error rate is very high it is between 0.8 to 0.9. Whereas LSTM produces a low RMSE value.

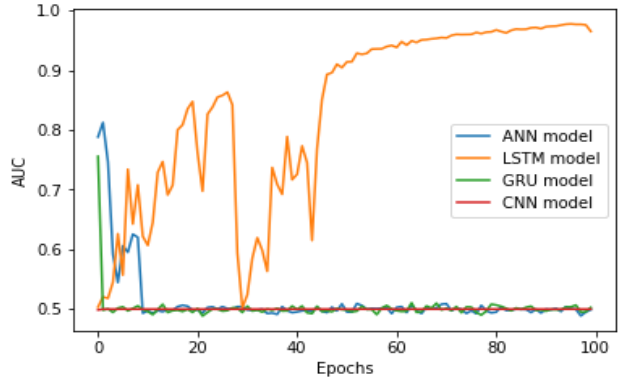

Figure 7. AUC

The Area Under the Curve (AUC) measures a classifier's ability to differentiate between classes, and it is used as a summary of the Receiver Operating Characteristics (ROC) curve. The more the AUC, the better the user sees the model's ability to differentiate between the positive and negative classifications.

Here Figure 7 represents AUC for the classification of Android malware, the graph compares the LSTM, ANN, CNN, and GRU models.

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + FN}$$

AUC is a better indicator of classifier performance than accuracy because it is unaffected by the size of the test or the assessment data. The proposed model's AUC is near 1.0 because LSTM is better at handling text data. Whereas existing models fail in handling text data and classification of malware data.

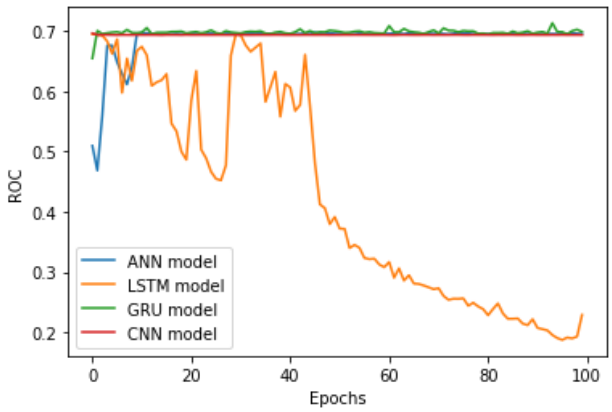

Figure 8. ROC

The relationship between TPR and FPR at various categorization levels is depicted by a ROC curve. As the classification threshold is lowered, more items are categorized as positive, resulting in a higher number of False Positives and True Positives in the system.

Here Figure 8 represents the ROC of the proposed LSTM and existing ANN, CNN, and GRU models. ROC of LSTM is good while an increasing number of epochs. But existing models' ROC is varying between 0.5 to 0.2. because ineffective of existing models in the classification of text data.

6. CONCLUSIONS

The unusual growth in Android malware over the last couple of years necessitates developing a more effective way to classify Android malware. This study proposes a deep neural network-based LSTM model for classifying Android malware. LSTM works well with text data and extracts effective features from the data for the classification of malware data. Whereas existing models are not effective in extracting text data. Our research examines the different existing systems that are presently utilized to identify malware and other malware activities and made comparisons with existing models ANN, CNN, and GRU models. The proposed LSTM model enhances the efficiency of the malware classification model by achieving an accuracy of more than 98% whereas existing models are not able to achieve 90% accuracy.

REFERENCE

[1] Mobile Threat Report 2020. https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf, accessed on Mar. 10 2021.

[2] Kouliaridis, V., Potha, N., Kambourakis, G. (2021). Improving android malware detection through dimensionality reduction techniques. Machine Learning for Networking, pp. 57-72. http://dx.doi.org/10.1007/978-3-030-70866-5_4

[3] Bacci, A., Bartoli, A., Martinelli, F., Medvet, E., Mercaldo, F., Visaggio, C. (2018). Impact of code obfuscation on android malware detection based on static and dynamic analysis. 4th International Conference on Information Systems Security and Privacy, pp. 379-385. http://dx.doi.org/10.5220/0006642503790385

[4] Kouliaridis, V., Kambourakis, G., Geneiatakis, D., Potha, N. (2020). Two anatomists are better than one-dual-level android malware detection. Symmetry, 12(7): 1128. http://dx.doi.org/10.3390/sym12071128

[5] Petsas, T., Voyatzis, G., Athanasopoulos, E., Polychronakis, M., Ioannidis, S. (2014). Rage against the virtual machine. In Proceedings of the Seventh EuropeanWorkshop on System Security—EuroSec '14. New York, USA.

[6] Roy, S., DeLoach, J., Li, Y., Herndon, N., Caragea, D., Ou, X., Ranganath, V., Li, H., Guevara, N. (2015). Experimental Study with Real-World Data for Android App Security Analysis Using Machine Learning. Proceedings of the 31st Annual Computer Security Applications Conference, ACSAC 2015, Los Angeles, CA, USA, pp. 81-90.

[7] Yan, P., Yan, Z. (2017). A survey on dynamic mobile malware detection. Softw. Qual. J., 26: 891–919. https://doi.org/10.1007/s11219-017-9368-4

[8] Odusami, M., Abayomi-Alli, O., Misra, S., Shobayo, O., Damasevicius, R., Maskeliunas, R. (2018). Android malware detection: A Survey. In Communications in Computer and Information Science. Springer International Publishing: New York, NY, USA.

[9] Kouliaridis, V., Barmpatsalou, K., Kambourakis, G., Chen, S. (2020). A survey on mobile malware detection techniques. IEICE Trans. Inf. Syst., E103-D(2):204-211. http://dx.doi.org/10.1587/transinf.2019INI0003

[10] Liu, K., Xu, S., Xu, G., Zhang, M., Sun, D., Liu, H. (2020). A review of android malware detection approaches based on machine learning. IEEE Access, 8: 124579-124607. https://doi.org/10.1109/ACCESS.2020.3006143

[11] Gibert, D., Mateu, C., Planes, J. (2020). The rise of machine learning for detecting and classifying malware: Research developments, trends, and challenges. J. Netw. Comput. Appl., 153: 102526. https://doi.org/10.1016/j.jnca.2019.102526

[12] Shabtai, A., Tenenboim-Checking, L., Mimran, D., Rokach, L., Shapira, B., Elovici, Y. (2014). Mobile malware detection through analysis of deviations in application network behavior. Comput. Secure., 43: 1-18. https://doi.org/10.1016/j.cose.2014.02.009

[13] Canfora, G., Mercaldo, F., Visaggio, C.A. (2015). Mobile Malware Detection using Op-code Frequency Histograms. In Proceedings of the 12th International Conference on Security and Cryptography, SCITEPRESS—Science and Technology Publications, Colmar, France. http://dx.doi.org/10.5220/0005537800270038

[14] Jang, J., Kang, H., Woo, J., Mohaisen, A., Kim, H.K. (2015). Andro-AutoPsy: Anti-malware system based on

similarity matching of malware and malware creator-centric information. Digit. Investig., 14: 17-35. http://dx.doi.org/10.1016/j.diin.2015.06.002
[15] Virusshare. https://virusshare.com/, accessed on Sept. 10 2020.
[16] Contagio. http://contagiominidump.blogspot.com/, accessed on Sept. 10 2020.
[17] Google Play. https://play.google.com/, accessed on Sept. 10 2020.
[18] Yerima, S.Y., Sezer, S., Muttik, I. (2015). High accuracy android malware detection using ensemble learning. IET Inf. Secure., 9: 313-320. http://dx.doi.org/10.1049/iet-ifs.2014.0099
[19] Coronado-De-Alba, L.D., Rodríguez-Mota, A., Escamilla-Ambrosio, P.J. (2021). Feature selection and ensemble of classifiers for Android malware detection. In Proceedings of the 2016 8th IEEE Latin-American Conference on Communications (LATINCOM), Medellin, Colombia, pp. 1–6.
[20] Arp, D., Spreitzenbarth, M., Huebner, M., Gascon, H., Rieck, K. (2014). Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket. Proceedings of the 21th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 12: 1128. http://dx.doi.org/10.14722/ndss.2014.23247
[21] Milosevic, N., Dehghantanha, A., Choo, K.K.R. (2017). Machine learning aided Android malware classification. Comput. Electr. Eng., 61: 266-274. https://doi.org/10.1016/j.compeleceng.2017.02.013
[22] Damshenas, M., Dehghantanha, A., Choo, K.K., Mahmud, R. (2015). M0Droid: An Android Behavioral-Based Malware Detection Model. J. Inf. Priv. Secure., 11(3): 141-157. http://dx.doi.org/10.1080/15536548.2015.1073510
[23] Idrees, F., Rajarajan, M., Conti, M., Chen, T.M., Rahulamathavan, Y. (2017). PIndroid: A novel Android malware detection system using ensemble learning methods. Comput. Secure., 68: 36-46. https://doi.org/10.1016/j.cose.2017.03.011
[24] Li, D., Li, Q., Ye, Y., Xu, S. (2021). A framework for enhancing deep neural networks against malicious malware. IEEE Transactions on Network Science and Engineering, 8(1): 736-750. https://doi.org/10.1109/TNSE.2021.3051354
[25] Athiwaratkun, B. Stokes, J.W. (2017). Malware classification with lstm and gru language models and a character-level CNN. 2017 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pp. 2482-2486.
[26] Gopi, A.P., Naik, K.J. (2021). A Model for Analysis of IoT based Aquarium Water Quality Data using CNN Model. In 2021 International Conference on Decision Aid Sciences and Application (DASA), pp. 976-980.
[27] Gibert, D.C. Mateu, J. (2019). Planes A hierarchical convolutional neural network for malware classification. The International Joint Conference on Neural Networks 2019, IEEE, pp. 1-8.
[28] Narayana, V.L., Gopi, A. (2021). Secure communication in Internet of things based on packet analysis. In Machine Intelligence and Soft Computing. Springer, Singapore.