JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 3657-3668
https://publishoa.com
ISSN: 1309-3452

Malware Detection of Android Based Systems using SIMGRU

B S N Murthy, Assoc. Prof ,CSE:BVCE, Email:murthy2007.b@gmail.com

P B V Rajarao, Assoc. Prof ,CSE:BVCE, Email:rajaraopbv@gmail.com

Divvi Likhitasri,CSE:BVCE

Dodda Pavani,CSE:BVCE

Doddi Sravan Sai Arun Kumar,CSE:BVCE

Maddukuri Venkata Durga Sai Jagadeesh,CSE:BVCE

Received 2022 March 25; **Revised** 2022 April 28; **Accepted** 2022 May 15.

ABSTRACT

To put it simply, malware is posing a serious threat to global network security as the Internet era has progressed at an unprecedented rate. Using SIMGRU, we present an Android malware detection method that falls under the static detection category. Using the clustering similarity, which is widely used in static analysis of Android malware, we were able to improve the Gated Recurrent Unit (GRU) and produce three distinct structures of Sim GRU: Input Sim GRU, Hidden Sim GRU, and Input hidden SIMGNU There are two types of input hidden sim gru: input and hidden. Results show that the GRU model and other methods are outperformed by the Sim GRU, Hidden GRU, and Input Hidden GRU.

Keyword: SimGRU (Gated Recurrent Unit)

I. INTRODUCTION:

Mobile software is constantly being released into the market, which makes it easier for malicious software to spread. This has led to a rise in the number of malicious Android apps and newer Android malware. [1] In order to evade detection and thwart analysis, Android malware is becoming increasingly sophisticated and powerful. As a result, it can be concluded that Android malware on mobile phones continues to pose a threat to device security. Malware can be identified in a variety of ways, as experts have proposed [2–9]. Static detection, dynamic detection, and hybrid detection are the three main types. An application's malicious code can be detected using static analysis, which reverse engineers the programme and extracts key features. It is possible to detect known malware quickly and effectively using static analysis, which is based on matching analysis. While the application is running in a simulated environment, malicious behaviour can be detected using the dynamic detection approach. Dynamic and static detection are both used in the hybrid detection method.

1.1 GRU

Among other things, GRU is an RNN model that eliminates the gradient disappearance issue common to native RNNs. A reset and an update gate are two of the GRU's two structural gates. An adaptive structure of two gates can be learned from the hidden state of the previous GRU unit's inputs and output.

1.2 The SIMGRU

We use GRU's similarity function, which is used by other malware detection systems, to enhance Android's malware detection performance. This model is referred to as SIMGRU.

1.3 Purpose

The purpose of this section is to explain the rationale for my decision to New techniques are being developed by malicious programmers in order to evade detection as all existing techniques are heavily reliant on static and malware analysis, where static analysis will extract dynamic features such as permissions and API calls, then check whether an

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 3657-3668
https://publishoa.com
ISSN: 1309-3452
app is requesting harmful permission or not, and if it is requesting harmful permissions, then that application will be flagged as malicious.

1.4 Problem statemdent:

For the first time, the author of this paper is using a similarity function called Euclidean Distance with GRU algorithm to extract similar features from a dataset, and this similar features will then be trained with GRU algorithm to improve the detection process. GRU neurons will be able to remember and predict similar neurons with a higher degree of accuracy, just like human brain neurons can remember similar events.

1.5 The Scope:

The static detection approach, which uses SIMGRU to detect Android malware, is what we propose.

The scope of the project is defined as follows:

Using GRU algorithms, we can find similarities between input and hidden neurons to gather more important features and then remove or hide the unimportant ones, increasing the accuracy of our predictions. GRU algorithms work on both input and hidden neurons. Because all malware-infected Android apps use the same API calls and other activities, identifying features that are similar allows us to extrapolate critical information that aids in our analysis.

1.6 SVM-based malware detection on Android smartphones using a vector of keywords

Smartphones have led to an increase in the number of mobile phone malwares, particularly on popular mobile operating systems such as Android, which can potentially harm users' personal information. It's been a challenge, however, to find a way to effectively detect new malware and malicious software variants. This paper presents a method for extracting features from Java source code in contrast to the traditional feature extraction method based on binary code. Key codes like API calls, Android permissions, the common parameters, and common key words in Android malware source code can be correlated using the Keywords Correlation Distance method. Afterwards, SVM is used to increase the system's ability to detect new malicious software as well as existing malware samples. Contrary to more traditional approaches, which rely on the text's context, this approach disregards that information. This method records the malicious software's behaviour by combining the characteristics of the various types of malicious software with the operating system itself. The method has been shown to be efficient and effective in detecting malware on the Android platform in tests.

Selection of representative samples by frequent subgraph analysis for the classification of Android malware families

Anti-malware systems face major challenges due to the rapid increase in Android malware, as the sheer number of samples overwhelms malware analysis systems. In order to speed up malware detection and inspection, malware samples should be classified into families, so that the common features shared by malware samples from the same family can be exploited. The selection of representative malware samples from each family can significantly reduce the number of malware samples that must be analysed. ' There are, however, a number of limitations to current classification solutions. For starters, the majority of Android malware is created by inserting malicious components into popular apps, which could lead to classification algorithms being misled. Second, Android malware that is polymorphic uses transformation attacks to avoid detection. To represent the common behaviours of malware samples belonging to the same family, we present in this paper a novel approach that creates frequent subgraphs (fregraphs). The FalDroid system we're proposing and developing classifies Android malware and selects representative samples based on fregraphs automatically. On 8407 malware samples from 36 families, we put it to the test. FalDroid can correctly classify 94.2 percent of malware samples into their families in about 4.6 seconds per app, according to experimental results. Malware investigation costs can be drastically reduced by using FalDroid, which only selects 8.5% to 22.5% of all samples that exhibit the most common malignant behaviour.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 3657-3668
https://publishoa.com
ISSN: 1309-3452

II. Method of detecting Android malware using the Naive Bayes and permission correlation algorithms

To better detect Android malware, an improved naive Bayes classification model for Android malware detection was proposed. An improved naive Bayes algorithm for malware detection is proposed in the first place, taking into account the unknown permissions that may be malicious in detection samples. Given the limited training samples, limited access to resources, and new malicious permissions in test samples, we weighted the new malware permissions in the test samples. The Android malware detection efficiency is enhanced by the weighted naive Bayesian algorithm. Second, we proposed a detection model for permissions and information theory based on the improved naive Bayes algorithm, taking into account the detection model. We looked at the relationship between the permission and the data. After determining the Pearson correlation coefficient r's value, we deleted permissions with r's value less than the threshold and obtained the new permissions set by deleting those permissions. So, using information theory, we were able to improve our detection model by clustering. Finally, in the same simulation environment, we found 1725 pieces of Android malware and 945 pieces of non-malicious code. An increased detection rate of 97.59 percent for benign applications is achieved using the improved naive Bayes algorithm. False detection rates are reduced by 8.25 percent using the improved naive Bayes algorithm.

4. Deep4MalDroid: A Linux kernel system call graph-based deep learning framework for Android malware detection,

Android malware detection is a hot topic in cyber security because of the explosive growth of Android malware and the harm it causes to smartphone users (e.g., the theft of user credentials, resource abuse). Anti-malware software products, such as Norton, Lookout, and Comodo Mobile Security, are currently the most significant line of defence against Android malware. The use of repackaging and obfuscation techniques by malware attackers to get around signatures and thwart attempts to analyse their inner mechanisms is on the rise. It is necessary to develop defences that can withstand the ever-evolving sophistication of Android malware, which necessitates the development of new techniques. Component Traversal is a novel dynamic analysis method that can automatically run the code routines of any given Android application (app) as thoroughly as possible. A deep learning framework based on graph-based features is used to detect newly discovered Android malware based on the extracted Linux kernel system calls. In order to compare various malware detection approaches, an extensive experimental study was conducted on a real sample collection from the Comodo Cloud Security Center. Experiments show that our method outperforms other Android malware detection techniques, which is encouraging. An anti-malware system for Android developed by us, Deep4MalDroid, has been integrated into a commercial product.

As a result of this work, the UCI ML repository's banknote authentication dataset was subjected to three different train test ratios: 80/20, 60/40, and 70/30. Attributes in the dataset include 1372 for features and 5 for the target attribute, which has a value as either genuine bank currency or fake currency.

III. Proposed Systems

3.1 GRU Algorithm

A similarity function called Euclidean Distance is used in this paper to extract similar features from a dataset, and these features are then trained using the GRU algorithm.

Using GRU algorithms, we can find similarities between input and hidden neurons to gather more important features and then remove or hide the unimportant ones, increasing the accuracy of our predictions. GRU algorithms work on both input and hidden neurons.

3.1.1. GRU, INPUT SIMGRU, HIDDEN SIMGRU, and INPUT HIDDEN SIMGRU were used in this study.

The performance of similarity functions in malware detection, which we call SimGRU, allows GRU to detect Android malware.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 3657-3668
https://publishoa.com
ISSN: 1309-3452

3.1.2 ASSERT SIMGRU

GRU uses a vectorized representation of the original data as its input vector xt. S=sim can be used to represent the similarity function (xt-1, xt). x=(1-s)xt-1 can be used to determine the degree of similarity between two points in time. Because of this, we've added a similarity function called InputSimGRU to GRU.

3.1.3. Hiding Simgru

Human neurons are modelled in GRU's hidden state ht, so the representation of GRU cells may improve Android malware detection. A similarity between two hidden states can be calculated as h=1-sht-2, and the similarity between two adjacent hidden states is s=ht-1sim. Known as HiddenSimGRU, this is the name of the GRU being proposed.

3.1.4 INPUT SIMGRU HIDEOUT

A GRU cell can learn more similarities by comparing input and hidden state similarity simultaneously. This is possible because the input and hidden state can be used to detect Android malware from different perspectives. Input Hidden Sim GRU is the name of the model.

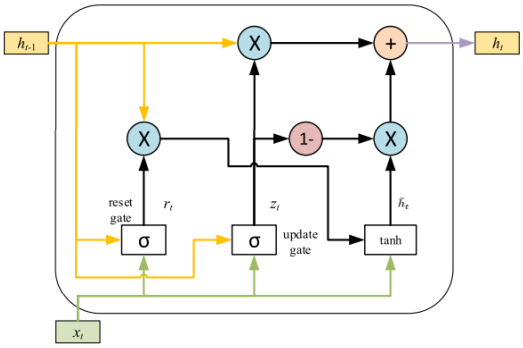

Fig.1. Architecture diagram

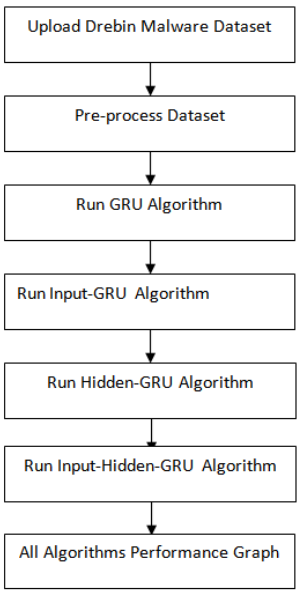

Fig. 2. Algorithm and Process Design diagram

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 3657-3668
https://publishoa.com
ISSN: 1309-3452
Use this module to upload the Drebin malware dataset and then view the total number of malicious and benign records in a graph.

Pre-process Dataset: This module will read the entire dataset and remove all of the non-existent values, before dividing the dataset into two separate training and testing sets.

Use this module to train the GRU algorithm on the above dataset without using any similarity functions, and then determine the accuracy of the train validation.

This module is used to train GRU algorithm by finding similarity between input features X and X-1 using Similarity Euclidean distance function. 3) Run Input-GRU algorithm.

To train GRU, we'll use this module's Similarity Euclidean distance function to compare two sets of unobserved hidden features.

We will use this module to train the GRU algorithm by comparing the inputs X and X-1 and the hidden features they contain to each other using the Similarity Euclidean distance function.

Algorithm performance metrics, such as precision, recall, and accuracy will be graphed in this module.

IV. Outcome and Implementation

4.1 The collection of data

The Drebin dataset was used by the author to implement this project, and we will be using the same dataset to implement this paper. Using different static APIs and permission calls, the dataset column details are illustrated in the diagrams below.

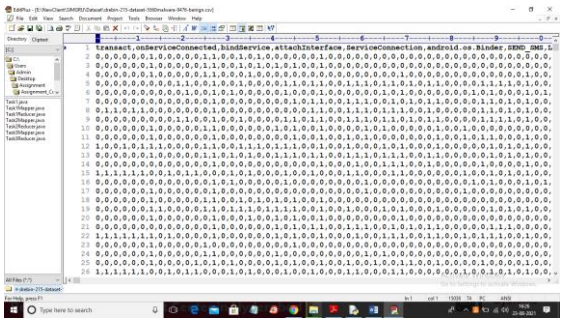

Fig.3. Dataset column

Android permission and API call names appear first in this dataset, followed by either 0 or 1 where 0 indicates that the app is not requesting permission or accessing that API, and 1 indicates a request for permission and/or an API call.

4.2 Metrics for Evaluation:

Performance, Accuracy, and Receiver Operating Capabilities-Area. We use ROC-AUC metrics to gauge how well our models are performing. FPR=False Positive Rate must be used to evaluate the F1-score, accuracy, precision, and recall.

To put it another way, TPR stands for True Positive Rate.

F1-score: Accuracy, Precision, Recall

True positive (TP) is defined as the number of events that are correctly identified.

An incorrectly predicted or unneeded number of events is referred to as a false negative (FN).

An FP is a number of events that were incorrectly predicted.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 3657-3668
https://publishoa.com
ISSN: 1309-3452
"True negative" (TN): the number of events that can be predicted correctly and are therefore unnecessary.

Ratio of falsely positive results: Machine learning model accuracy can be measured using this metric. https://deepchecks.com/glossary/machine-learning-model-accuracy The formula for calculating the FPR is: FP/(FP+TN)

a measure of the accuracy of a test result It is a synonym for recall and is therefore defined as TPR=FP/(FP+TN).

Accuracy: To measure performance, simply divide the number of correctly predicted observations by the total number of predicted observations.

Accuracy=(TN+TP)/(TP+FP+TN+FN)

It's the ratio that accurately predicts positive observations in the original data.

TP/(TP+FN) = Recall

In order to get the most accurate results, precision is needed. In other words, this means determining the total number of software's predicted to be positive that are actually positive. Precision is calculated as follows: TP/(TP + FP) = TP/(TP).

F1-score: The F-score is a way of combining precision and recall in a machine learning model. high levels of accuracy and recall https://deepai.org/machine-learning-glossary-and-terms/precision-and-recall Mean

Precision and recall are two important properties of the model, and they are both described by the term "harmonic mean" at https://deepai.org. The F-score is another name for it. Precision Recall/Precision + Recall is the formula used to calculate F1 Score.

If you're trying to solve a classification problem, you'll want to keep an eye out for metrics that can help you determine how well you're doing.**4.3 Outcome:**

Fig.4. Upload Drebin Malware Dataset diagram

In above diagram click on ‘Upload Drebin Malware Dataset’ button to upload dataset

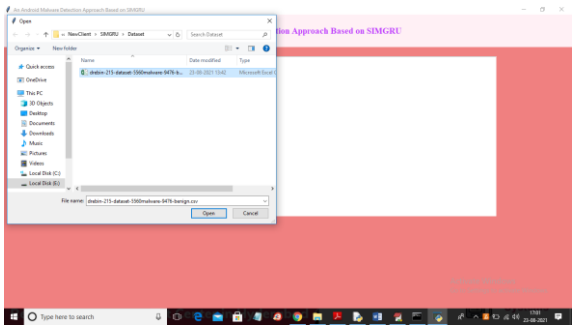

Fig5. Upload Upload Drebin Malware Dataset

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 3657-3668
https://publishoa.com
ISSN: 1309-3452

In above diagram selecting and uploading ‘drebin’ dataset and then click on ‘Open’ button to load dataset and to get below diagram

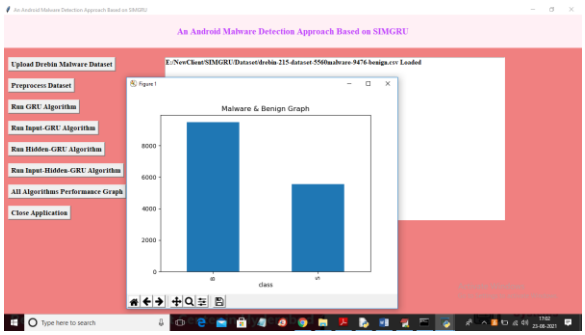

Fig.6. number of malware and benign (normal) records

In above diagram in text area we can see dataset loaded and in dataset application find total number of malware and benign (normal) records. In above graph x-axis represents type of record where B means benign and S mean malware and y-axis represents count of those records. Now close above graph and then click on ‘Preprocess Dataset’ button to process dataset by removing missing values and to get below diagram

Fig. 7. Dataset Processing

In above diagram we can see dataset process and in dataset total 15036 records are there and each record contains 215 features or values. Now dataset is ready and now click on ‘Run GRU Algorithm’ button to train GRU with above dataset and to get below output

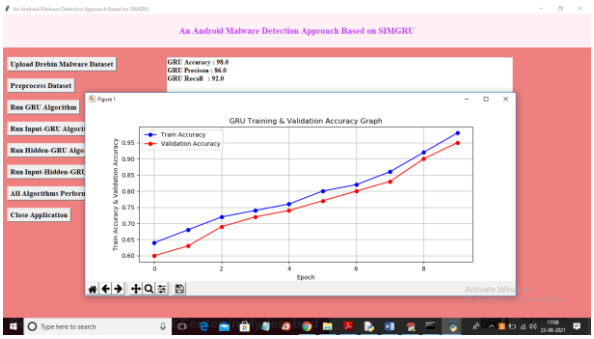

Fig.8. GRU accuracy, precision and recall

In above diagram we can see GRU accuracy, precision and recall values and we in graph we can see TRAIN and validation accuracy of GRU. To train GRU we took 10 EPOCHS and in above graph x-axis represents EPOCH and y-axis represents accuracy values. In above graph blue line represents training accuracy and red line represents validation

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 3657-3668
https://publishoa.com
ISSN: 1309-3452

accuracy. Now close above graph and then click on 'Run Input-GRU Algorithm' button to get below output

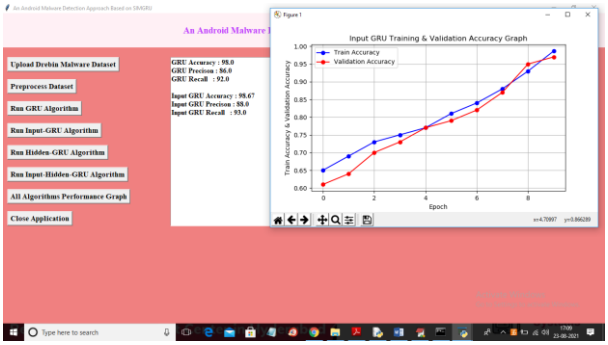

Fig. 9 Train and Validation Accuracy

In above diagram we can see Input-GRU output and now close above graph and then click on 'Run Hidden-GRU Algorithm' button to get below output

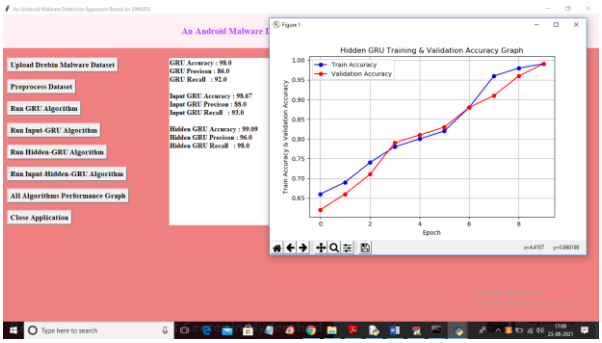

Fig. 10 Hidden-GRU output

In above diagram we can see Hidden-GRU output and now close above graph and then click on 'Run Input-Hidden-GRU Algorithm button to get below output

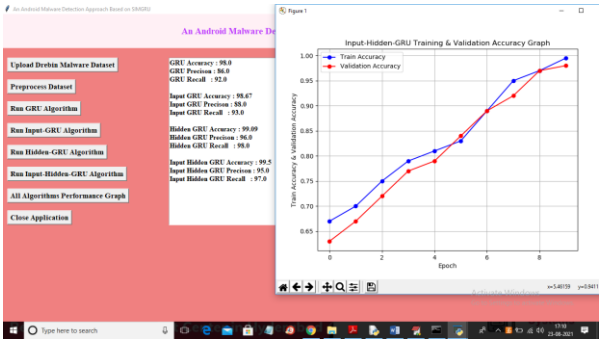

Fig. 11. Input-Hidden-GRU' output

In above diagram we can see 'Input-Hidden-GRU' output and now in above output we can see Input-Hidden-GRU got high accuracy compare to other algorithms and now close above graph and then click on 'All Algorithms Performance Graph' button to get below graph.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 3657-3668
https://publishoa.com
ISSN: 1309-3452

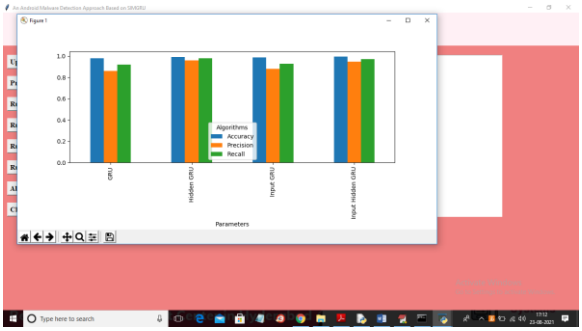

Fig. 12. Input-Hidden-GRU

In above graph x-axis represents algorithm names with different bars representing different metrics and y-axis represents metric values. In above graph Input-Hidden-GRU has got high values compare to other algorithms. From above graph we can conclude that 'Input-Hidden-GRU' is giving better result

Extension Outcomes:

In this paper we have introduced GRU algorithm with various versions like HIDDEN and INPUT HIDDEN and getting accuracy up to 99.50% and in extension we have added Convolution2D neural network with multiple filters and 4 dimensional data as input features which allow CNN to optimize features with much more flexibility with more filters and multi dimension data. This optimize features helps in improving accuracy. As extension we have added two algorithms called CNN to further optimize accuracy and PREDICTION module to detect malware or normal records from Test data

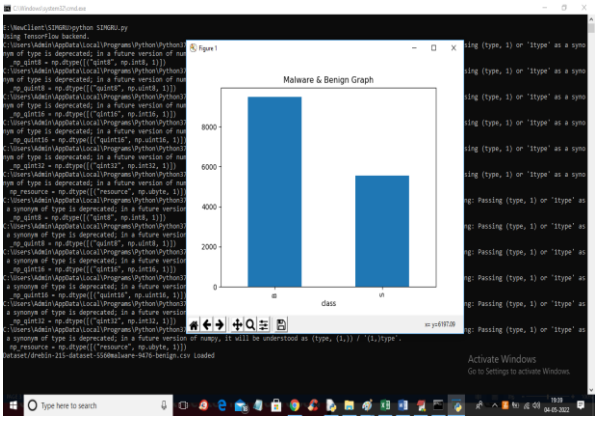

Fig. 13 Malware and normal records

In above graph getting number of malware and normal records and now close above graph to get below output

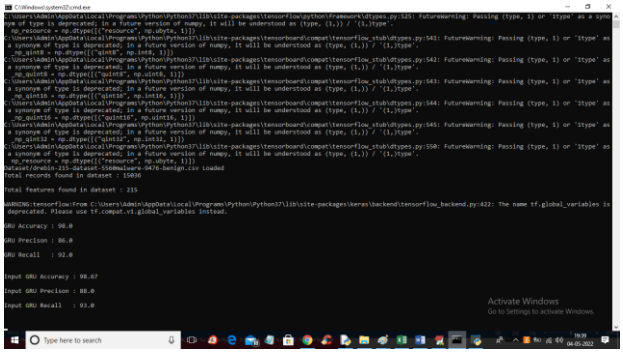

Fig. 14.GRU Accuracy

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 3657-3668
https://publishoa.com
ISSN: 1309-3452

In above diagram with GRU we got 98% accuracy and with Input GRU we got 98.67 accuracy and now close graph to get below output

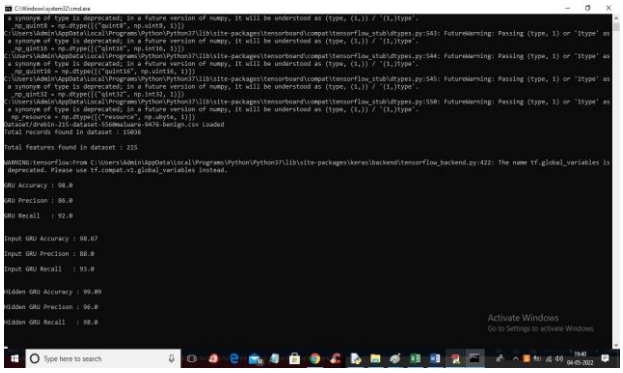

Fig. 15. Hidden GRU

In above diagram with Hidden GRU we got 99.09 accuracy and close graph to get below output

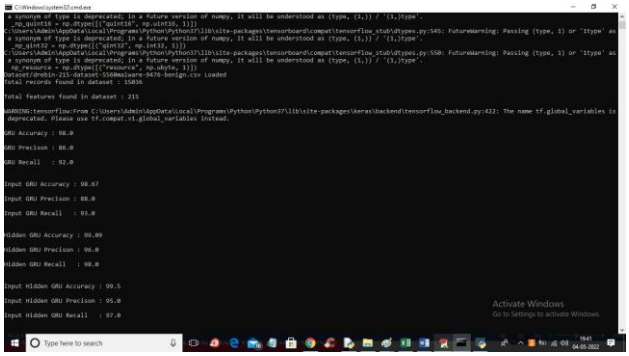

Fig.16. Input Hidden GRU

In above diagram with Input Hidden GRU we got 99.50% accuracy and close graph to get below output

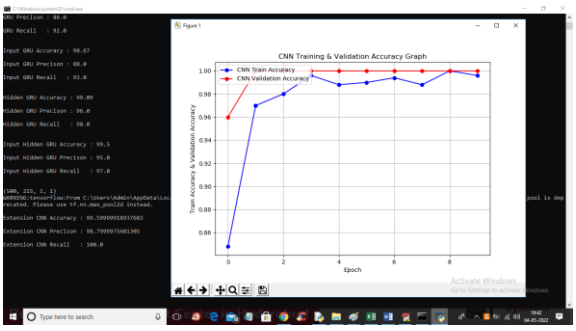

Fig. 17 Extension CNN

In above diagram with Extension CNN we got 99.59% accuracy and recall as 100% and this value is more than other algorithms and in above training graph we can see CNN validation and training accuracy reached to 100% and now close above graph to get below graph.

JOURNAL OF ALGEBRAIC STATISTICS
Volume 13, No. 2, 2022, p. 3657-3668
https://publishoa.com
ISSN: 1309-3452

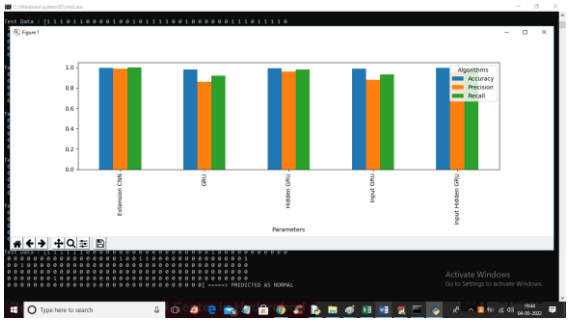

Fig. 18. CNN Extension

In above graph x-axis represents algorithm names and y-axis represents accuracy and other metrics in different colour bar and in above graph we can see CNN extension got high accuracy and RECALL values compare to other algorithms and now close above graph to get below output.

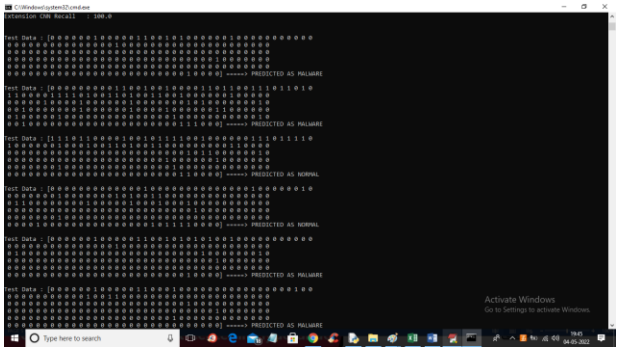

Fig. 19 CNN prediction

In above diagram in square bracket you can see the TEST data values and after =➔ arrow symbol you can see CNN prediction as NORMAL or MALWARE for each record

CONCLUSION

This paper introduces the GRU algorithm with various versions such as HIDDEN and INPUT HIDDEN and achieves an accuracy of up to 99.50%, and in an extension we have added the Convolution2D neural network with multiple filters and 4 dimensional data as input features, which allows CNN to optimise features with much more flexibility with multiple filters and multi-dimensional data. As a result of these enhanced features, accuracy has been improved.

Additional CNN algorithms have been added to improve the accuracy and prediction of detecting malware or normal records from test data.

Bibliography

[1] F. Jaafar, G. Singh, and V. Zavarsky, ‘‘An analysis of Android malware behavior,’’ in Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. Companion, Lisbon, Portugal, Jul. 2018, pp. 337–341.

[2] J. Sun, K. Yan, X. Liu, C. Yang, and Y. Fu, ‘‘Malware detection on Android smartphones using keywords vector and SVM,’’ in Proc. IEEE/ACIS 16th Int. Conf. Comput. Inf. Sci. (ICIS), Wuhan, China, May 2017, pp. 833–838.

[3] C. Wang, Q. Xu, X. Lin, and S. Liu, ‘‘Research on data mining of permissions mode for Android malware detection,’’ Cluster Comput., vol. 22, no. S6, pp. 13337–13350, Nov. 2019.

[4] K. A. Talha, D. I. Alper, and C. Aydin, ‘‘APK auditor: Permission-based Android malware detection system,’’ Digit. Invest., vol. 13, pp. 1–14, Jun. 2015.

[5] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, and T. Liu, ‘‘Android malware familial classification and representative sample selection via frequent subgraph analysis,’’ IEEE Trans. Inf. Forensics Security, vol. 13, no. 8, pp. 1890–1905, Aug. 2018.

[6] F. Shang, Y. Li, X. Deng, and D. He, ‘‘Android malware detection method based on Naive Bayes and permission correlation algorithm,’’ Cluster Comput., vol. 21, no. 8, pp. 1–12, 2017.

[7] K. Allix, T. F. Bissyandé, Q. Jérome, J. Klein, and Y. Le Traon, ‘‘Empirical assessment of machine learning-based malware detectors for Android,’’ Empirical Softw. Eng., vol. 21, no. 1, pp. 183–211, 2016.

[8] S. Wu, P. Wang, X. Li, and Y. Zhang, ‘‘Effective detection of Android malware based on the usage of data flow APIs and machine learning,’’ Inf. Softw. Technol., vol. 75, pp. 17–25, Jul. 2016.

[9] S. Hou, A. Saas, L. Chen, and Y. Ye, ‘‘Deep4MalDroid: A deep learning framework for Android malware detection based on Linux kernel system call graphs,’’ in Proc. IEEE/WIC/ACM Int. Conf. Web Intell. Workshops (WIW), Omaha, NE, USA, Oct. 2016, pp. 104–111.

[10] M. Junaid, D. Liu, and D. Kung, ‘‘Dexteroid: Detecting malicious behaviors in Android apps using reverse-engineered life cycle models,’’ Comput. Secur., vol. 59, pp. 92–117, Jun. 2016.

[11] Y. Aafer, W. Du, and H. Yin, ‘‘DroidAPIMiner: Mining API-level features for robust malware detection in Android,’’ in Proc. Int. Conf. Secur. Privacy Commun. Syst., Sep. 2013, pp. 86–103.

[12] M. Fan, J. Liu, W. Wang, H. Li, Z. Tian, and T. Liu, ‘‘DAPASA: Detecting Android piggybacked apps through sensitive subgraph analysis,’’ IEEE Trans. Inf. Forensics Security, vol. 12, no. 8, pp. 1772–1785, Aug. 2017.

[13] X. Wang, W. Wang, Y. He, J. Liu, Z. Han, and X. Zhang, ‘‘Characterizing Android apps’ behavior for effective detection of malapps at large scale,’’ Future Gener. Comput. Syst., vol. 75, pp. 30–45, Oct. 2017.

[14] J. Li, Z. Wang, T. Wang, J. Tang, Y. Yang, and Y. Zhou, ‘‘An Android malware detection system based on feature fusion,’’ Chin. J. Electron., vol. 27, no. 6, pp. 1206–1213, Nov. 2018.

[15] S. Shang, N. Zheng, J. Xu, M. Xu, and H. Zhang, ‘‘Detecting malware variants via function-call graph similarity,’’ in Proc. 5th Int. Conf. Malicious Unwanted Softw., Oct. 2010, pp. 113–120.

[16] M. Xu, L. Wu, S. Qi, J. Xu, H. Zhang, Y. Ren, and N. Zheng, ‘‘A similarity metric method of obfuscated malware using function-call graph,’’ J. Comput. Virol. Hacking Techn., vol. 9, no. 1, pp. 35–47, Feb. 2013.

[17] A. Utku and I. A. Dogru, ‘‘Malware detection system based on machine learning methods for Android operating systems,’’ in Proc. 25th Signal Process. Commun. Appl. Conf. (SIU), Antalya, Turkey, May 2017, pp. 1–4.

[18] J. Song, C. Han, K. Wang, J. Zhao, R. Ranjan, and L. Wang, ‘‘An integrated static detection and analysis framework for Android,’’ Pervas. Mobile Comput., vol. 32, pp. 15–25, Oct. 2016.

[19] S. Rastogi, K. Bhushan, and B. B. Gupta, ‘‘Android applications repackaging detection techniques for smartphone devices,’’ Procedia Comput. Sci., vol. 78, pp. 26–32, Mar. 2016.

[20] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio ‘‘Learning phrase representations using RNN encoder-decoder for statistical machine translation,’’ 2014, arXiv:1406.1078. [Online]. Available: https://arxiv.org/abs/1406.1078