International Journal of Core Engineering & Management
Volume-7, Issue-08, 2023 ISSN No: 2348-9510

# DETECTING MALWARE ATTACKS BASED ON MACHINE LEARNING TECHNIQUES FOR IMPROVE CYBERSECURITY

*Sanjeet Kumar Choudhary*
*Birla Institute of Technology,*
*Ranchi, India*
*sanjeet.choudhary1@gmail.com*

*Piyush Ranjan*
*AVP Barclays, NJ USA*
*piyushranjangc@gmail.com*

*Sumit Dahiya*
*Apeejay College of Engineering,*
*Gurgaon , India*
*sumitdahiya1234@gmail.com*

*Sidharth Kumar Singh*
*New York University, NY USA*
*sks592@nyu.edu*

*Abstract*

*As Android continues to gain more market share as the mobile operating system and was at 85 % of smartphones in 2020 the issue of cybersecurity or rather android malware becomes a top issue. In this research, different ML techniques for Android malware detection are analyzed using a large dataset from 2008 to 2012 encompassing 9,476 benign and 5,560 malicious applications belonging to 179 different families. Extra Trees, CNN, CNN-LSTM, SVM, LSTM are the study's classification models to compare in terms of efficiency and effectiveness by analyzing the F1-score, recall, accuracy, and precision metrics. Preprocessing was done which includes Label encoding, Normalization and Feature scaling for the dataset. This research shows that while conventional methods like examining the signature of malware fail with the constantly changing malware, using anomaly-based detection using ML proves useful. With a maximum accuracy of 99.92%, the results demonstrate that CNN outperforms the other models. This demonstrates the CNN model's superior capability in detecting and classifying malicious Android applications. The research highlights the importance of adopting advanced ML techniques to address the dynamic and sophisticated nature of Android malware, providing valuable insights for enhancing malware detection and cybersecurity strategies in the Android ecosystem. The results highlight the need of constantly improving detection systems to stay up with the changing threat environment.*

*Keywords: Android Malware, Cyber Security Strategies, Android Ecosystem, Drebin Dataset, Malicious Behavior, CNN.*

International Journal of Core Engineering & Management
Volume-7, Issue-08, 2023 ISSN No: 2348-9510

## I. INTRODUCTION

There has to be satisfactory resolutions to several concerns related to cybersecurity, which has emerged as a major subject of urgent concern for computer scientists and network engineers. The rapid advancement of technology and its natural integration into every aspect of our life has led to a comprehensive examination and recognition of a broad variety of malware applications and their intended targets. Malware targeting Android devices has been making waves as of late and is well-known for the amount of attention it garners online. Android is now dominating the operating system industry and is one of the most popular operating systems overall. 85% of smartphones using Android as their preferred operating system as of 2020 were powered by this technology.

A vast array of apps are often pre-installed on an Android device. The number of these applications surpassed 2.8 million by the end of April 2020. Most Android applications may be accessed on Google Store, where you can get a selection of these apps. Malware penetration has become more likely due to the system's friendliness and the increasing reliance on it by app developers and users [1][2]. This results from the harmful programs' inherent ability to carry out their functions smoothly and with the help of several important components. A few of them include the environment in which an Android program runs, the amount of permissions permitted, and invoking third-party code.

Malware intrusive tactics are always evolving to avoid detection. Some malware apps include over 50 variables, making it very difficult to identify them. Consequently, given the constant growth of Android malware, it is crucial to devise ways that can identify, disable, and eliminate it efficiently. The researchers in the area were indeed preoccupied by all of these challenges, which motivated them to conduct research in order to identify malware and address it in a proper manner[3][4].

The Android operating system is currently using a number of techniques to differentiate between legitimate and malicious apps. Without installing and using the application, these strategies provide a priori insights. An use of ML techniques to malware detection and malware family behavior prediction has garnered significant attention from cybersecurity specialists in recent years [5][6]. Solutions based on ML outperform those relying on signatures when it comes to freshly released malware. A more accurate representation of features may be achieved via the application of DL algorithms that are capable of autonomous feature engineering [7]. By evaluating several ML models on the Drebin dataset, this study aims to enhance the effectiveness of Android malware detection. With Android holding a sizable portion of the market and malware becoming more sophisticated, it is imperative to provide reliable detection techniques that can quickly recognize and categorize both new and established dangerous apps. To determine the most accurate and dependable method for differentiating between safe and harmful applications, the study will assess models including CNN, CNN-LSTM, LSTM, Extra Trees, and SVM. The final objective is to strengthen malware detection skills and offer insights into how various models perform according to F1-score, precision, accuracy, and recall, therefore supporting improved cybersecurity measures within the Android ecosystem. The primary findings of the study on Android Security with Malware Detection, which made use of the Drebin dataset, are as follows:

- The study utilized the Drebin dataset, comprising 9,476 benign and 5,560 malicious samples from 179 distinct Android malware families, to provide a thorough evaluation of malware detection methods and contribute valuable insights to the field of Android security.
- By comparing a range of machine learning models, including Extra Trees, CNN, CNN-LSTM, SVM, and LSTM, the study identified the LSTM model as the most effective

according to F1-score, recall, accuracy, and precision, offering a robust approach to Android malware detection.

- The research included comprehensive performance measurements, including as F1-score, recall, precision, and accuracy; these metrics were shown graphically using tables and graphs, which made it easier to comprehend the advantages and disadvantages of each model.
- A study emphasized the value of data preparation methods, showing how label encoding, normalization, and feature scaling improve the accuracy and dependability of ML models in malware detection.
- By examining the performance of many models, the study helped to clarify the advantages and disadvantages of various methods for detecting malware, which will direct future investigations and advancements in the area of Android security.

### 1. Structure of the paper

This study is organized as follows: The methods for identifying malicious apps on Android are discussed in Section II. In Section III, we detail the methodology used for the research. We present and discuss the outcomes of the experiment in Section IV. Find the findings and recommendations for future studies in Section V.

## II. LITERATURE REVIEW

This section discusses a number of literature-based initiatives pertaining to Android malware detection. The most important publications for static, dynamic, and hybrid malware analysis are summarized in the section's sequel.

In this research, M. Al-Janabi and A. M. Altamimi (2020) shows how to accurately categories malware into nine distinct families. To classify the malware applications, various techniques are used, including RF, DT, SVM, KNN, SGD, LR, NB, and deep learning. With an accuracy of 96%, the result demonstrates that the DL model performs better than other comparable ML techniques[8].

In this study, Trang Nguyen, Tho Nguyen and Vu, (2021) provide a fresh approach to prototype extraction for ML classification that deviates from the conventional model. The distances between malware samples and prototypes will determine their classification. Our technique achieves an F1 score of 97.7 percent for unknown malware classification and 96.5 percent for known malware classification in the testing findings, demonstrating its superior performance over the previous prototype-based classification method[9].

In this paper, Cohen, Nissim and Elovici, (2020) introducing MalJPEG, the first ML system designed expressly to efficiently identify unknown harmful JPEG images. We ran a comprehensive MalJPEG test on 156,818 photographs taken in the real world; 155,013 (98.85%) were safe, while 1,805 (1.15%) were harmful. Pairing MalJPEG with the LightGBM classifier yields the greatest detection results, with an AUC of 0.997, a TPR of 0.951, and a very low FPR of 0.004.[10].

This paper, S. Khalid and F. B. Hussain, (2022) presents an extensive study of malware detection using ML techniques. In this paper, Malware detection techniques applied on the dataset CCCS-CIC-AndMal-2020 and get the accuracy of 99.48%. Table 1 shows the comparative research studies on android security with ML for effective malware detection[11].

International Journal of Core Engineering & Management
Volume-7, Issue-08, 2023 ISSN No: 2348-9510

**TABLE I. Comparative Research Of Android Security With Machine Learning For Effective Malware Detection**

| Ref | Methodology | Dataset | Result | Limitation |
|---|---|---|---|---|
| [8] | Classified malware into 9 families using ML and DL models. | DREBIN | DL achieved 96% accuracy. | May not handle novel malware well. |
| [9] | Prototype-based classification with hyper cuboids. | Malgenome | F1 score: 96.5% (known), 97.7% (unknown). | Performance on complex malware is unclear. |
| [10] | MalJPEG uses features from JPEGs with LightGBM. | Virus Share, AndroZoo | AUC: 0.997, TPR: 0.951, FPR: 0.004. | Limited to JPEG images only. |
| [11] | ML techniques with ensemble learning and feature selection. | CCCS-CIC-AndMal-2020 | Accuracy: 99.48%. | May not adapt well to evolving malware threats. |

### 1. Research gaps

Android virus detection using machine learning has yielded encouraging results with high accuracy rates. Adapting these models to new malware threats is a major research need. Most studies use known malware families and datasets, which may not completely represent the continually changing environment of malware, including new, undetected, or highly disguised versions. While individual models like deep learning and ensemble approaches work well in controlled settings, their scalability in real-world scenarios, especially with unbalanced or insufficient data, remains unknown. Contextual or behavioral malware factors that might improve detection beyond static or syntactic analysis are also understudied. To safeguard Android devices from future dangers, these shortcomings could pave the way for more efficient and adaptable malware detection systems.

## III. RESEARCH METHODOLOGY

The goal of this research is to create a ML–based malware detection system that works. The methodology for this study involves several key steps to ensure robust analysis and effective malware detection using the Drebin dataset. Data pre-processing included label encoding categorical variables, normalization of numerical features, and feature scaling to standardize the data. Next, the dataset was divided into sets for testing, validation, and training. To evaluate their effectiveness in identifying malware, a number of classification models were used, including CNN, Extra Trees, CNN-LSTM, SVM, and LSTM. Performance was evaluated employing a variety of metrics, including as confusion matrices, recall, accuracy, and precision. The findings were presented in a graphical and tabular format to allow for a thorough evaluation of the performance of each model. Figure 1 depicts the flowchart of the method for android security with malware detection.

International Journal of Core Engineering & Management
Volume-7, Issue-08, 2023 ISSN No: 2348-9510

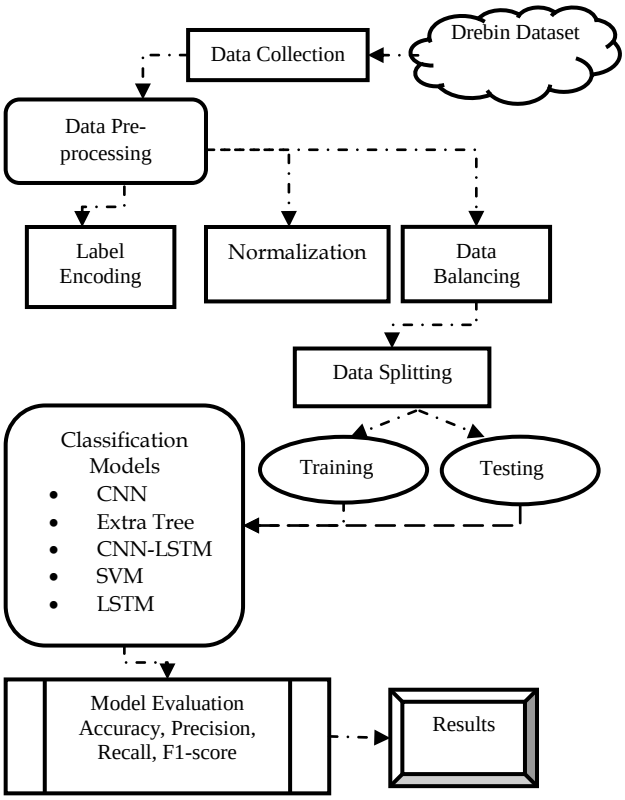


Fig. 1. Methodology diagram for the malware detection system

The following flowchart steps are listed. Each level of data processing in the system is explained in depth.

**1. Data collection**

For this study, it involves a total of 9,476 benign samples and 5,560 malicious samples, representing 179 distinct Android malware families. These samples were collected over a period from 2008 to 2012, providing a comprehensive dataset for detailed analysis.

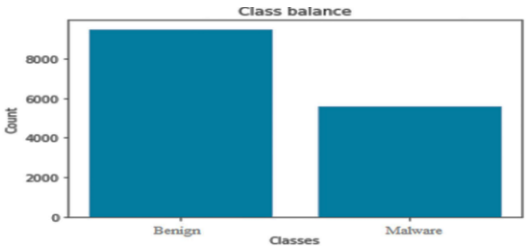


Fig. 2. Count plot of dataset classes

International Journal of Core Engineering & Management
Volume-7, Issue-08, 2023 ISSN No: 2348-9510

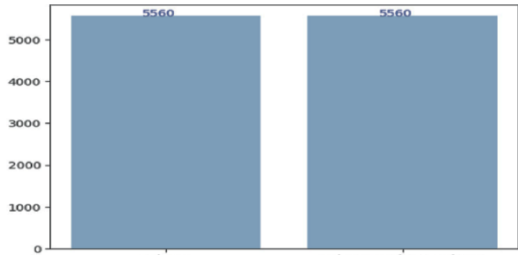

Fig. 3. Count plot of dataset classes with SMOTE

The following Figure 2 and 3 shows the count plot of data classes with imbalance problem and balance with SMOTE of Drebin data.

**2. Data preprocessing**

The phrase "data preparation" describes the repeated steps used to convert unstructured data into a more manageable form. Raw datasets are characterized by mistakes, incompleteness, inconsistencies, lack of behavior, and patterns [12]. The following are the fundamental pre-processing techniques:

**3. Label Encoding**

Many algorithms are unable to directly interpret categorical variables, hence converting them to numerical representation is necessary to make them suitable for use in ML models. Label encoding is a common technique used for this transformation. In this process, each unique category is assigned a distinct numerical value. For example, in our dataset, the categorical items "B" and "S" in the class column are encoded as 0 and 1, respectively. Specifically, "B" (benign) is represented as 0 with 9,476 occurrences, while "S" (malicious) is encoded as 1 with 5,560 occurrences.

**4. Normalization**

The process of normalizing numerical data involves bringing it into line with a standard range, usually ranging from 0 to 1, so that every characteristic makes an equal contribution to the model's performance. This technique helps in improving the convergence of algorithms and prevents features with larger values from disproportionately influencing the results.

**5. Data Balancing**

The dataset may exhibit class imbalance as a result of the likelihood that there are much fewer malware samples than benign ones. Several tactics were used to address this problem, including under sampling, oversampling, and creating fake data for the minority class.

6. **Data Splitting**:

The first step in developing a machine learning model is to partition the dataset into three parts: training, validation, and testing. While the training and testing sets are used for training the model, the validation and testing sets are employed for hyperparameter tuning.

**7. Classification Model**

This section focuses on the comparative analysis of different classification models applied to the Drebin dataset. By evaluating these models against the dataset's features, the aim is to assess and

International Journal of Core Engineering & Management
Volume-7, Issue-08, 2023 ISSN No: 2348-9510

contrast their performance. The objective is to identify which model delivers the most precise and valuable insights for analyzing customer feedback and product ratings.

**1) CNN**

Create a CNN architecture that can classify textual input while considering the data's sequential structure. Features may be extracted using a convolutional layer, the sample size can be reduced by pooling, and classification can be accomplished through fully connected layers [13]. Employ suitable normalization methods and activation functions such as ReLU.

A. **Convolution Layer:** As a foundational element of CNN, the convolution layer functions as its building block. Based on the dataset, these layers, which are made up of kernels of size two or three, execute convolution at an input and forward a resulting output to a layer behind it. The integral values of two functions are measured in this layer [14]. The filter, width, and stride characteristics may be specified as inputs in the Python implementation due to the use of the "conv1d" function. The utilized kernel sizes were 1 and 2. Furthermore, the Gaussian distribution was used to initialize the filters and biases. Subsequently, a pooling layer was implemented, into which the convolution layer's output is sent as input. For this layer, Eq 1 is the formula that we applied in our implementation.

$$C = \frac{(I + 2P - F)}{S} + 1 \dots\dots\dots\dots (1)$$

To clarify, "I" stands for input, "P" for padding, "F" for filter, and "S" for stride.

B. **Fully connected layer:** Every neuron in the network is connected to every neurone in the layer below it via this layer. The multiplication of the bias effect matrix is crucial to the functioning of a network that uses convolutional, fully connected, and hidden layers. In particular, the fully connected layers are crucial for the input-output mapping. Consider a two-layer structure that makes use of weights to establish connections between them. These masses, which represent the interconnected layers of a network. According to Eq. 2, this layer functions as a constant in the system and includes a bias function.

$$V^k = r_k^{FL}(V^{k-1}) = \left((V^{k-1})\right)w_i^k + b^k \dots\dots\dots\dots (2)$$

C. **Activation function**: An essential part of neural networks is the activation function, which adds non-linearity to an input signals and sends its output to a next layer to be used by neurons. There were a number of activation functions to choose from; we settled on ReLU and Leaky ReLU. ReLU outperforms more conventional functions like tanh and sigmoid in terms of computing speed and efficiency due to its capacity to selectively stimulate a subset of neurons. One-way ReLU gets rid of negative values in the activation map is by making them zero.

$$f(x) = \begin{cases} 0, & for\ x < 0 \\ x, & for\ x \geq 0 \end{cases} \dots\dots\dots\dots (3)$$

$$f(x_i) = \begin{cases} a_i x_i, & for\ x < 0 \\ x_i, & for\ x \geq 0 \end{cases} \dots\dots\dots\dots (4)$$

Equation 3 defines the mathematical expression of ReLU while Equation 4 defines LReLU. Results of convolution calculations in the convolutional layer are represented by both 'a' and 'b' in these equations. The hyper-parameter 'b' may be adjusted to get the best possible results; we trained

International Journal of Core Engineering & Management
Volume-7, Issue-08, 2023 ISSN No: 2348-9510

with a modest value for it. Since the convolution layer used ReLU, calculation was rapid and training time was minimized; the next layer used LReLU for processing.

Pooling layer: A CNN layer's dimensionality may be decreased by using the pooling layer. Its main function, which is to reduce the network's computational cost and minimize its size by limiting overfitting, is to be positioned between convolutional layers. In this layer, the functions employed are max and average pooling. The process of adding a max-filter to the input results in max-pooling, which essentially reduces the size of the network by down sampling it. Eq. 5 provides an illustration of the formula used in this layer.

$$P_{ot} = \frac{(I_d - F_d)}{S} + 1 \dots\dots\dots\dots (5)$$

The pooling layer dimension is denoted as I_d the filter dimension as F_d and the stride as S.

**8. Model training and testing**

The textual data that made up the training dataset was used to train the model. There are 12024 features in the training feature set that are used in the process. Given that the objective includes binary classification (malware vs. benign), use an appropriate loss function, like binary cross-entropy. Adam and RMSprop are two optimization algorithms that may be used to optimize the model's parameters. To identify the optimal configuration for the CNN architecture, tune the hyperparameters using the validation dataset. Use a test dataset of 3007 features to evaluate the built model. In both the training and testing phases, experiment with various filter and pool sizes, layer counts, learning rates, and batch sizes.

## IV. RESULT ANALYSIS AND DISCUSSION

The section on results provides an overview of the model's findings along with some commentary on them. All of the models utilized in this study were implemented in Python, and a number of supportive libraries, such as NumPy, pandas, matplotlib, and sklearn, were used for result evaluation. Using a machine with 16 GB of RAM and a Core i7, fifth-generation CPU, the applicable model was fitted.

**1. Performance Measures**

This section lays out the performance measurements and formulae used to compare the different techniques in this study and evaluates the models' respective performances. The following are the parameters:

**A. Confusion Matrix**

The confusion matrix, often known as the error matrix, is fundamentally used in statistical categorization. Algorithm performance may be seen via a certain table layout. In this matrix, each row stands for a possible value prediction and each column for the actual value; or vice versa. A total of four cells—"true positive," "true negative," "false positive," and "true negative"—make up the output matrix. TP indicates a positive relationship among the actual and predicted values, whereas TN shows a positive relationship among the actual and the model's predicted value. Conversely, FP denotes a negative actual value but a positive model projected value; FN, on the other hand, denotes a negative combination of both predicted and actual values. The following performance measures as discussed below:

International Journal of Core Engineering & Management
Volume-7, Issue-08, 2023 ISSN No: 2348-9510

### B. Accuracy

Taking an average of the numbers along the main diagonal will give you a good idea of the matrix's accuracy. It is given as in Equation (6):

$$Acuracy = \frac{TP + TN}{TP + TN + FN + FP} \ldots . (6)$$

### C. Precision

This metric is determined by dividing the overall number of positive outcomes by a number of outcomes accurately forecasted by a classifier. It is expressed as in Equation (7):

$$Precision = \frac{TP}{TP + FP} \ldots . (7)$$

### D. Recall

A measure of accuracy is a ratio of a number of appropriate samples to a number of correct positive results. This may be expressed mathematically as Equation (8):

$$Recall = \frac{TP}{TP + FN} \ldots . (8)$$

### E. F1-score

It serves as a benchmark for test accuracy. Recall and precision harmonic means are represented by the F1 Score. The F1 Score is between 0 and 1. It gives you details on your classifier's precision and resilience. It may be expressed mathematically as follows in Equation (9):

$$F1 - score = 2 * \frac{precision * recall}{precision + recall} \ldots . (9)$$

## 2. Experiment Results

The experimental outcomes of CNN machine learning model on the Drebin Dataset are shown in this section. Figures, graphs, and tables are used to illustrate the results and give a thorough summary of the models' capabilities. The Table 2 shows the performance of CNN model for malware detection with evaluate measures that get 99% and 98% detection rate

**TABLE II. CNN MODEL PERFORMANCE FOR MALWARE DETECTION.**

| Models | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|
| CNN | 99.92 | 98.61 | 99.16 | 98.88 |

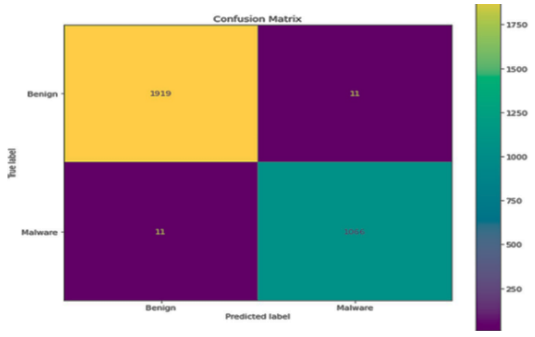


Fig. 4. Confusion matrix of CNN model on Drebin dataset.

International Journal of Core Engineering & Management
Volume-7, Issue-08, 2023 ISSN No: 2348-9510

Figure 4 represents a CNN model's confusion matrix on the Drebin dataset. The two labels are shown in the table with "forecast label" on a x-axis and "True label" on a y-axis. "Benign" is the label on the x-axis and "Malware" is the label on the y-axis. The cell displays the frequency of each category; for example, cell 1919 in the top left, cell 11 in the top right, cell 10 in the bottom left, and cell 1066 in the bottom right all show the frequency of occurrences.

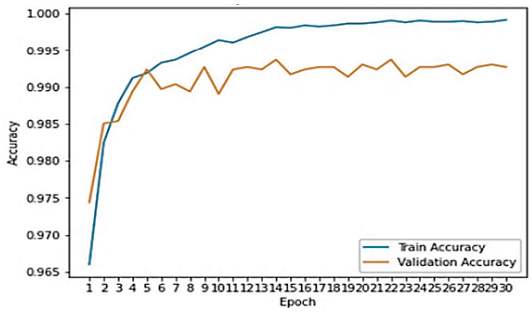


Fig. 5. CNN plot explanation of training and validation accuracy.

The correlation between time and train loss is seen in Figure 5. The x-axis indicates the epoch, while the y-axis shows the precision. The model's accuracy during validation was 99.92%, whereas it was 100% during training. The accuracy during validation is shown by the orange line, whereas the accuracy during training is shown by the blue line. There has been an upward trend in validation accuracy and a downward trend in train correctness. This suggests that the model is becoming too used to the training data and is failing to properly incorporate fresh data as the epoch count increases. The model's oversimplification and over-fitting to the training set are the causes of its poor performance on new data.

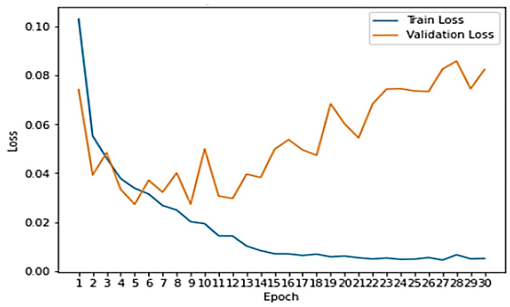


Fig. 6. CNN plot explanation of training and validation loss.

Figure 6 displays a visual depiction of validation and training loss. We can see "Train Loss" on an orange line and "Validation Loss" on a blue line. Here, we see an essential metric "Loss" represented by its values on the vertical axis and a continuous variable "Epoch" portrayed on the horizontal axis. An orange line is about 0.08 in length and terminates at 0.09 in length. The blue line finishes at around 0.06 and begins at approximately 0.04. Although there are occasional variations, both lines typically show an increasing tendency.

### 3. Comparative analysis

Table 3 compares many ML models for malware detection using the Drebin dataset, accounting for factors such as recall, accuracy, precision, and f1-score.

International Journal of Core Engineering & Management
Volume-7, Issue-08, 2023 ISSN No: 2348-9510

TABLE III. Comparison between various ml model for malware detection.

| Models | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|
| CNN | 99.92 | 98.61 | 99.16 | 98.88 |
| Extra Trees [15] | 87.75 | 89.35 | 85.33 | 0.97 |
| CNN-LSTM [16] | 97.20 | 97.72 | 97.92 | 97.82 |
| SVM [17] | 95.21 | 0.98 | 0.97 | 0.98 |

The findings of evaluating several ML models that were used to detect malware from the Drebin dataset are shown in Table 3. The CNN model outperformed all others with remarkable results: an F1-score of 98.88%, a recall of 99.16%, a precision of 98.61%, and an accuracy of 99.92% when it came to detecting harmful and benign samples. Even the CNN-LSTM model provided a good number of results with 97.20% balanced accuracy and F1-score of 97.82% which exhibits the model on how effective it is on sequential data. On the other hand, Extra Trees classifier showed poor performance with accuracy of 87.75% and a significant difference between precision of 89.35% and recall of 85.33% resulting in very low F1 score0.97. On the T2 dataset, the proposed model showed reasonable accuracy of 95.21%, precision0.98 and recall0.97, thereby having F1- score0.98. In detail, the CNN output outcompeted the other models, though Extra Tree placed the lowest output, which negatively impacted recall thus hurting its ability to detect the malware.

## V. CONCLUSION AND FUTURE WORK

The study used the Drebin dataset to assess several ML models for Android malware detection in depth. The models that were examined include CNN, Extra Trees, CNN-LSTM, SVM, LSTM and CNN of which CNN model had the highest accuracy, precision, recall, and F1-score at 99. 92%, 98. 61%, 99. 15%, and 98. 88%, respectively. This exceptional performance shows that CNN has a robust ability in separating between the benign and the malicious applications. Conversely, the Extra Trees model had the worst results in all the metrics that were evaluated, which indicates the model's poor detection capability of Android malware in contrast with other models. The outcome shows that the new approaches using machine learning especially CNN has higher accuracy and reliability in contrast to traditional approaches like signature-based detection.

Future studies should concentrate on the following areas to increase our understanding of how to identify malware on Android devices. First of all, there is a possibility that a combination of CNN's strength with the strength of other methods of machine learning will perform even better. Furthermore, one can improve feature extraction techniques as well as increase the scope of the dataset to cover the more recent malware samples to enhance the models' resilience and coverage against new threats. Further, analysis of the role of different hyperparameters and potential utilization of transfer learning techniques could also provide answers to increasing the detection accuracy rates. Last of all, the inclusion of real-time malware detection and improving an interpretability of the ML models for Android benefits the end-users as it would make Android a safer environment for mobile users.

**REFERENCES**

1. G. Suarez-Tangil and G. Stringhini, "Eight Years of Rider Measurement in the Android Malware Ecosystem," IEEE Trans. Dependable Secur. Comput., 2022, doi: 10.1109/TDSC.2020.2982635.
2. V. K. Yarlagadda and R. Pydipalli, "Secure Programming with SAS: Mitigating Risks and Protecting Data Integrity," Eng. Int., vol. 6, no. 2, pp. 211–222, Dec. 2018, doi: 10.18034/ei.v6i2.709.
3. A. S. Shatnawi, A. Jaradat, T. B. Yaseen, E. Taqieddin, M. Al-Ayyoub, and D. Mustafa, "An Android Malware Detection Leveraging Machine Learning," Wirel. Commun. Mob. Comput., 2022, doi: 10.1155/2022/1830201.
4. C. Wressnegger, K. Freeman, F. Yamaguchi, and K. Rieck, "Automatically inferring malware signatures for anti-virus assisted attacks," in ASIA CCS 2017 - Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security, 2017. doi: 10.1145/3052973.3053002.
5. N. Milosevic, A. Dehghantanha, and K. K. R. Choo, "Machine learning aided Android malware classification," Comput. Electr. Eng., 2017, doi: 10.1016/j.compeleceng.2017.02.013.
6. M. S. Akhtar and T. Feng, "Malware Analysis and Detection Using Machine Learning Algorithms," Symmetry (Basel)., 2022, doi: 10.3390/sym14112304.
7. G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, 2013. doi: 10.1109/ICASSP.2013.6638293.
8. M. Al-Janabi and A. M. Altamimi, "A comparative analysis of machine learning techniques for classification and detection of malware," in Proceedings - 2020 21st International Arab Conference on Information Technology, ACIT 2020, 2020. doi: 10.1109/ACIT50332.2020.9300081.
9. T. T. Trang Nguyen, D. Tho Nguyen, and D. L. Vu, "A Hypercuboid-Based Machine Learning Algorithm for Malware Classification," in Proceedings - 2021 RIVF International Conference on Computing and Communication Technologies, RIVF 2021, 2021. doi: 10.1109/RIVF51545.2021.9642093.
10. A. Cohen, N. Nissim, and Y. Elovici, "MalJPEG: Machine Learning Based Solution for the Detection of Malicious JPEG Images," IEEE Access, 2020, doi: 10.1109/ACCESS.2020.2969022.
11. S. Khalid and F. B. Hussain, "Evaluating Dynamic Analysis Features for Android Malware Categorization," in 2022 International Wireless Communications and Mobile Computing, IWCMC 2022, 2022. doi: 10.1109/IWCMC55113.2022.9824225.
12. D. Tanasa and B. Trousse, "Advanced Data Preprocessing for Intersites Web Usage Mining," IEEE Intell. Syst., 2004, doi: 10.1109/MIS.2004.1274912.
13. V. Rohilla, S. Chakraborty, and R. Kumar, "Deep learning based feature extraction and a bidirectional hybrid optimized model for location based advertising," Multimed. Tools Appl., 2022, doi: 10.1007/s11042-022-12457-3.
14. V. Rohilla, S. Chakraborty, and M. Kaur, "An Empirical Framework for Recommendation-based Location Services Using Deep Learning," Eng. Technol. Appl. Sci. Res., 2022, doi: 10.48084/etasr.5126.
15. M. Abuthawabeh and K. Mahmoud, "Enhanced android malware detection and family classification, using conversation-level network traffic features," Int. Arab J. Inf. Technol., 2020, doi: 10.34028/iajit/17/4A/4.

16. H. Alkahtani and T. H. H. Aldhyani, "Artificial Intelligence Algorithms for Malware Detection in Android-Operated Mobile Devices," Sensors, 2022, doi: 10.3390/s22062268.
17. P. Raghuvanshi and J. P. Singh, "Android Malware Detection Using Machine Learning Techniques," in Proceedings - 2022 International Conference on Computational Science and Computational Intelligence, CSCI 2022, 2022. doi: 10.1109/CSCI58124.2022.00200.