JCBI
Journal of Computing & Biomedical Informatics
ISSN: 2710 - 1606
Volume 07 Issue 01
2024

Research Article
https://doi.org/10.56979/701/2024

A Deep Learning Framework Based on GCN Model for Android Malware Detection

Syeda Huma Zaidi[1*], Muhammad Fuzail[1], Ali Raza[1], Yasir Aziz[2], Muhammad Kamran Abid[1], and Naeem Aslam[1]

[1]NFC Institute of Engineering and Technology Multan, Multan, 66000, Pakistan.
[2]Department of Computer Engineering, Bahauddin Zakariya University, Multan, Pakistan.
[*]Corresponding Author: Syeda Huma Zaidi. Email: humazaidi729@gmail.com

Received: February 09, 2024 **Accepted:** May 08, 2024 **Published:** June 01, 2024

Abstract: Nowadays, Android malwares are increasingly significantly producing major security issues. The complexity and increase of malware threats have made automated malware detection research an important component of network security. Traditional malware detection methods include manual examination of every malware file present in the application, which consumes a significant number of human resources (on the basis of both storage and time). Additionally, malware developers have created methods like code obfuscation to get beyond antivirus companies' conventional signature-based detection methods. Deep learning (DL) approaches for malware detection are now being used to resolve this issue. In this study, Performance comparisons are made amongst GCN (Graph Convolutional Network) models for Android malware detection. Using graph-based representations of malware of the Android DEX file, a GCN-based model is suggested to detect Android malware. GCN extracts the necessary features from the images of malware. The static approach is used to extract the essential features. Then, these features train GCN to detect malware. We presented a GCNs latest version for modeling more advanced graphical semantics. It automatically discovers and understands semantic and ordered patterns based on the previous stage's vectors, without requiring additional sophisticated or expert features. The proposed method outperformed the compared models in every performance metric, achieving an accuracy of 99.69% compared to other approaches.

Keywords: Android Malware Detection; Deep Learning; Graph Convolution Network; Static Feature Extraction.

1. Introduction

Malware detection is becoming a significant concern due to large numbers and malware complexity. Malware detection relies on conventional techniques based on heuristics and signatures; sadly, these approaches have limited generalization to unknown attacks and are immediately defeated with obfuscation approaches. In recent years, by learning valuable representations from data, Machine Learning and especially Deep Learning have made remarkable achievements in malware detection and are now chosen over conventional approaches[1]. In the past few years, malware has become a much greater threat to both humans and businesses. As the complexity and number of malwares increases, detection techniques rely on heuristics and signatures have lost their effectiveness. As a major means to understand this issue, the machine learning methods used for detecting malware are receiving significant attention. In the area of malware research, Graph Neural Network (GNN) is one such machine learning technology that gained interest recently as an effective tool for capturing the structural correlations between aspects of malware samples[2, 3].

In this research, we propose an approach that leverages multiple Graph Convolutional Network (GCN), (which is one of the most popular and important GNNs) iterations through program interactions

to analyze and predict dangerous or non-malicious files[4]. The main objective is to highlight the benefits of GCNs over more traditional methods and demonstrate how they can be used to identify malware more effectively and efficiently.

GNNs are capable of processing graph-based data, where nodes represent individual items and edges show connections between the components[5]. They have shown to be effective in a variety of academic disciplines where data is represented as a graph, including Natural Language Processing, Computer Vision, and Analysis of Social Network.

A malicious software program is designed to harm computer systems, steal confidential data, or break into a network without authorization. Their growing sophistication and complexity make them hard to identify and assess, which puts computer systems and networks' security at serious risk. Malicious software, sometimes known as malware, is harmful software intended to harm computer equipment and secretly collect user information. The most frequent types of malware are viruses, worms, adware, ransomware, spyware, and Trojan horses[6, 7].

As per the Cyber security Ventures Report 2021, cybercrime would result in losses up to $10.5 trillion per year by 2025[8]. A cyber security firm found approximately 5,683,694 malware installation packages for mobile devices in 2020; 5,704,599 more malicious installation packages such as mobile malware, adware, and risk ware were found in 2023. Figure 1 shows the development of malware from 2020 to 2023. Additionally, cybercriminals employ a variety of obfuscation strategies to impede the detection of malware by conventional signature-based methods. Furthermore, it's critical to detect new malware strains quickly since any delays might result in significant security breaches.

Android was the most prevalent operating system as of May 2021, with over 3 billion monthly active users. According to Forrester Research, it holds approximately 74% mobile market share, with iOS coming in second with a market share of 21%. It is projected that smartphone sales would rise by 7.7% in 2020, reaching a billion units by 2021. According to the World Retail Banking Report 2020, the COVID-19 epidemic has increased customer preference for online banking, and a sizable section of those consumers choose mobile banking apps. Mobile banking Trojans have increased dramatically in 2019 from 69,777 to 156,710 in 2020[9]. This information explains why malware detection is essential and why attackers are interested in banking data. We must detect malware as soon as possible because it is the most challenging work.

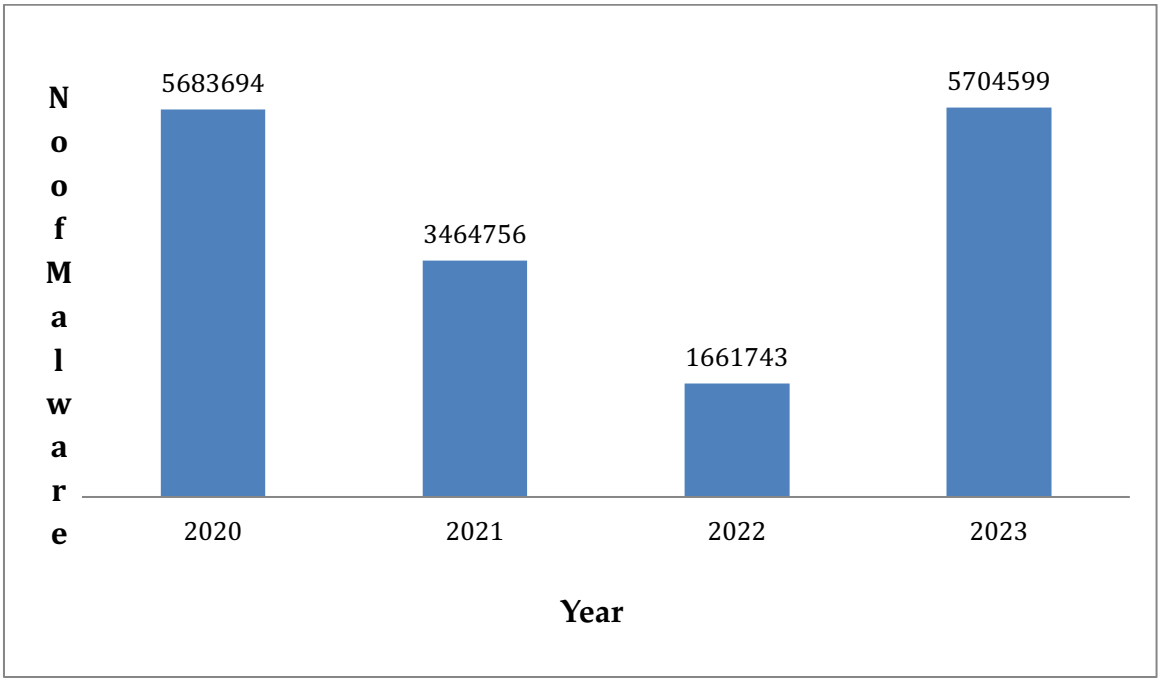

Figure 1. Malware attacks all past four years

Threats and malware are spreading daily, and many major information systems are being compromised, causing enormous economic losses. Personal information and data are crucial for every person. Everyone wants complete security in this era of hacking and stealing someone's personal information. In the current era, many approaches are used for detecting malware. For example,

CNN(Convolutional Neural Network), RNN(Recurrent Neural Network), DNN(Deep Neural Network), GAN(Generative Adversarial Network) and GCN(Graph Convolutional Network). Any organization will have procedures in place to stop and prevent attacks when the threat created by malware is significant. One of these preventive methods, which we will be talking about throughout this paper, is malware detection[10].

In this paper, we use GCN based static approach for malware detection. GCN is semi-supervised learning on structured data of Graph. It is based on an effective CNN variant that works directly with Graphs. GCN is a generated form of CNN in which the numbers of node interactions vary, and the nodes are not ordered. GCN does not share weights between hidden layers. A static approach extracts features and trains our GCN model to detect malware easily. GCN captures the behavior of malware adequately.

To make sure users are not vulnerable to threats, malware detection and mitigation is essential for malware detection. In malware analysis, the most popular methodologies are dynamic, static, and hybrid analysis methods. The static analysis looks at the files without actually running the application. Static analysis offers the best code coverage, fastest virus detection, and least amount of overhead; but, obfuscation techniques and dynamic code loading may slow it down. Because it collects data while the application is running in a sandbox, dynamic analysis is effective in handling code obfuscation. It does, however, come at a higher cost in terms of overhead and longer analytical times. Hybrid uses both static and dynamic approaches. It can run the code and also deal with code obfuscation[11, 12].

The two primary categories of malware detection techniques are behavior-based detection and signature-based detection. The signature- based detection method is used for detecting malware that looks for a program's code for particular signatures or patterns that are linked to malware. The approach Behavior-based detection identifies malware by examining its behavior and activities on a system[13]. But both approaches have some drawbacks that make them not suitable for the prevention of new malware attacks.

Convolution neural networks are used to detect image-based malware but nowadays graph-based malware techniques are used. When we train our CNN model it can extract high features. But direct convolution is not applied on graphs and it cannot detect graph-based malware. Even previously unknown malware can be detected using machine learning-based approaches, which also produced noticeably better detection performance. However, they rely heavily on feature engineering, which takes time and requires a special set of skills.

1.1. Research Question

Q: Which features are used to classify that application is Benign or not?

Q: How can we effectively measure the performance of our proposed approach?

Q: What is the working of higher-level graphical semantics?

1.2. Research Objective

The objective of this paper is to detect malware with the help of Deep learning (DL) approaches that GCN (a generalized version of CNN) recently took the role of more established methods for detecting Android malware. This paper's contributions are as follows:

1. We aim to detect malware in graph-based data. Direct convolution is not applied to graphs so we proposed a Graph convolution network.
2. Using GCN with the static approach so it can easily extract features.
3. These static features are used to detect whether the application is benign or not.
4. We compare different approaches and present new deep learning methods that detect malware efficiently.

To experimentally assess our framework's performance in distinguishing "Benign" from "malware" and locating the malware version, we show the attributes that it has learned.

1.3. Research Contribution

The contributions of this paper are mainly described below:

- We discuss deep learning methods and Graph convolution network that is used in our paper.
- We proposed GCN, a graph-based malware method that is used to detect malware with the help of graphs.
- The framework introduces the concept of NLP (Natural language Processing) word embedding technology. Word embedding technology maps all words, characters, and lexical information of

malicious features into vectors. Then these vectors illustrate the syntax and semantic connections between words that are situated next to one another in the space of vectors and share common information.

- A GCN model for high-level graphical semantics modeling was presented. Without the need for extra complex or expert features, it automatically finds and learns the semantic and ordered patterns utilizing vectors from the previous stage.

2. Literature Review

This section examines many conventional and current malware detection methods and provides a detailed explanation of the previous research on this topic. We will also talk about the advantages and disadvantages of these methods, as well as current developments in the domain, which will encourage us to suggest our graph-based malware detection strategies utilizing GCN and its various forms.

A string-matching rough method presented by Boyer Moore for malware detection with signatures. The strategy aims to improve detection efficiency by reducing the temporal complexity of signature matching[13]. Combine the data section of Android Manifest.xml files and DEX files to produce grayscale images. After that, a Temporal Convolutional Network (TCN) is used to scan the images for Android malware[14, 15].

They utilize the similar methodology here, but they only use the data section of the DEX files to obtain the image[16]. CNNs were also used to recognize images of Android malware produced by Hilbert space-filling curves with native instructions from mobile applications[17].

They create models for malware detection that are based on seven different types of static features including opcode, string, and API feature[18] [51].

Yen et al. employed CNNs to decompile Android APKs into code, then it is subsequently transformed to images through the TF-IDF approach[19]. A ResNet was trained on images obtained through the color visuals of Android app functions for detecting malware[20, 21].

Some studies have looked into additional deep learning methods to improve the efficiency of Android malware detection platforms. Zhang et al. developed a capsule network design, which replaces capsule layers with pooling layers in CNNs. Chimera learned features from visuals that are transformed from DEX files, data that is static like Android intents and access controls, and dynamic data like system call sequences, respectively, using multi-model deep learning that consists of a DNN, CNN, and TN[22, 23].

Emotion based reorganization using a graph neural network that is used to detect the emotion of anything, using large scale representation learning for Android malware detection[24]. Malware classification using CNN to detect android malware[25]. Android malware and group samples that are used to find families that are well-known [26].

Chen et al. and Darus et al. employed a model XGBoost to categorize the features collected from images of Android malware as harmful or non-malicious[27, 28]. Huang et al. introduced MixDroid, which uses several characteristics and machine learning classifiers merged via bagging to detect Android malware[29, 30].

For malware images, Gu et al. suggested using executable adversarial instances for machine learning classifiers. Most studies employed CNNs to detect Android malware. The superior performance of convolutional neural networks in computer vision applications is widely recognized. They can quickly pick up on the traits of malicious images and use them to classify malware that has been packed and unpacked[12, 31]. For example, the models that are CNN-based were trained on images retrieved from the Dalvik byte-code provided in the classes.Dex files of the android APK to detect android malware[32, 33]. They use the DBN method for graph representation[34] [49].

DySign thought that dynamic analysis may produce effective fingerprints for malware on Android devices. They have suggested a fingerprint method that protects from Android malware assaults by using dynamic analysis[35]. They detect malware by using graph representation learning. They take benign and malware samples from Drebin datasets and compare with our approach. They have the same accuracy but with some point difference as compared to our accuracy[25, 36]. Table 1 data shows how our model is giving accurate detection of malware as compared to other detection models.

Table 1. Comparing the suggested method with different approaches

Reference	Models	Number of Features	Malware Dataset	Benign Dataset	Accuracy
[32]	CNN using a feature layer of higher order	DEX bytecode features of images	Drebin dataset	Anzhi App store	95.10%
[37]	GCN	Graph feature extraction	Drebin, AMD, Malgenome	Google play store	92.30%
[38]	GCN	Graph based detection	AMGP, DB, AMD	Google play store	98.99%
[39]	GNN	IoT malware detection	CICMal Droid, Drebin	Androzoo(Drebin)	98.33%
[9]	EfficientNet-B4 CNN	Image features from byte-code	R2-D2 + other sources	R2-D2 Hsien-De Huang and Kao (2018)	95.7%
[40]	GCN	AI based malware detection	VirusTotal, VirusShare	System program and Internet	98.32%
[41]	Bi-LSTM+GNN	Call trace-based malware detection	VirusShare, Drebin, Droid Analytics	Google play store	97.69%
Proposed Approach	Effective GCN	graph-based feature extraction	AMD	AndroZoo	99.69%

3. Proposed Approach

3.1. Features Extraction

In malware detection, feature extraction is the process of locating and picking relevant characteristics, properties, or features that are obtained from raw data (such as system calls or binary code) that may be utilized to differentiate between malicious and non-malicious software. Then, by feeding the chosen characteristics into machine learning techniques, models that accurately categorize unknown data as benign or malicious may be trained. Below

Figure 2 shows the working of feature extraction and feature transformation

3.1.1. Graph Representation Learning

To identify Android malware, we offer a unique end-to-end Deep learning solution that builds a non-linear embedding based on the graph representation provided by static features. An Android application's main features are its rights, components, and application programming interfaces (APIs). These fundamental components of an Android application are the APIs. Arp et al. (2014) provided the methodology used to extract these APIs.

Every observable string was included in a text database that held all of the static features. Furthermore, we extracted several NLP features, words, characters, and lexical features from the text database as nodes for a graph using word embedding. We model every application of Android of graphs with node properties based on these NLP features. For every node in the network, our objective is to learn vector representation that is low-dimensional such that various types of vectors can be used to represent

different node attributes. We include joint-feature vectors for a malware sample in the hybrid model and assess it using several datasets. We can make complete use of a malware's grammatical and semantic information in this way.

3.1.2. Process for Feature Extraction

The goal of this research is to create an effective and dependable analysis method for identifying Android malware. With this approach, we may avoid the need for intricate code analysis and get optimal performance with minimal complexity. We were able to easily analyze the content of programs by extracting static features using the open-source technology Androguard2. Different features are extracted to find the features that are most suited for the classification algorithm in order to achieve the greatest performance. Furthermore, because the same parsing mechanism is used throughout training and testing, deep learning-based systems can successfully discover and classify problems.

Certain patterns and combinations of different traits are frequently indicative of malicious operations. The Android operating system is known for having a stringent access right management system. Malware usually requests a unique set of rights and frequently requests more access than benign software. In reality, the list of permissions requested can be used by some skilled antimalware developers to roughly detect malware. The permissions features are therefore essential. Furthermore, reusing code is a popular approach in app development. Various malwares within the same family frequently repurpose certain parts to carry out comparable malevolent acts. According to inconsistent naming of an app's components is one kind of significant attribute that could be helpful in recognizing well-known Android malware components.

Malware differs from innocent programs in the patterns and feature values it distributed. One of the most appealing ways to reveal a piece of code's harmful activity is through API call sequences. In order to contextualize our results, we have chosen a few dubious API call characteristics from published research on Android malware identification which could indicate harmful activity within an application or the coding practices of its creators. In actuality, even if their scripts might be obfuscated, various malwares from the same family are frequently employed with the same patterns to bring some sensitive APIs. We depend on information from suspicious API calls as a result it provides important semantics about the operation of the infection.

3.1.3. Lexical Vector

We specifically define the lexical aspects to better characterize the virus behavior. We regarded lexical knowledge as specifically learned word vectors, or "lexical vectors," designated by the letter L. There are two primary components to the lexical characteristics. The statistical data is presented in the first section. Upon examination of the previously extracted static properties, we deduced that malicious malware exhibits unique context structures distinct from those of benign applications. To determine the quantity of unique characters, and quantity of letters, the percentage of unique characters, and percentage of letters, we conduct a statistical study. There are 45 traits in all, all of which are combined to form Lexical 1.The details of these features that are based on lexical are described in Table 2. The variations in occurrence semantic characteristics like embedding of word and n-gram features are analogous to these statistics features.

Table 2. Lexical-based Features in Detail

Number of Feature	Numbers
Each letter's number, For example, "a," "b," "c," "d," "e," and "z,"	26
The number of each distinct character (like '-', '>', '(', ')', '/', '.', ';', '_')	8
The percentage of the largest continuous letter length in the static feature string	1
The proportion of unique characters in the string of static feature	1
The percentage of all letters that are most continuous letters in length	1
The largest consecutive unique character size as a proportion is found in the static feature string.	1
The percentage of all unique characters that have the greatest continuous length of any unique character	1
Number of letters	1
A lengthy letter that has been written consecutively	1
The quantity of unique characters	1

The longest continuous duration of a unique character	1
The length of the static feature string (including all letters and unique characters)	1
The portion of the static feature string's total letters	1

The second section discusses the malware's operations. 32 typical malware jobs were selected, as indicated in Table 3. For instance, The Context type (Landroid/content/Context) frequently provides worldwide application information (including unique assets, types, and resources) (Aafer et al., 2013).

Table 3. A description of the 32 different kinds of Android malware

1. Androids. Permissions. Vibrate	17. Ljava and lang and Process
2. Androids. Permissions. Call_Phone	18. Ljava abd util and Timer
3. Androids. Permissions. READ_LOGS	19. Ljava /lang/ Runtime
4. Android. Permission. Receive_Boot_Completed	20. Ljava and io and File Output Stream
5. Android. Permission. ACCESS_NETWORK_STATE	21. Androids .Hardwares .cameras
6. Androids. Permissions. WAKE_LOCK	22. Ljava /io /DataOutputStream
7. Landroid and OS and Bundle	23. Androids. Intents. categorys
8. Androids.Permissions.Read_Phone_State	24. Landroid /apps /Services
9. Landroid and Content and pm and PackageManager	25. Androids. Hardwares. telephonys
10. Landroid and Content and Context	26. Androids. Hardwares. locations.gps
11. Android.permission.INTERNET	27. Androids. Hardwares. screens.landscape
12. Landroid and telephony and TelephonyManager	28.Landroid /net /NetworkInfo
13. Androids. Permissions. Send_Sms	29. Androids. Hardwares. Locations. Networks
14. Landroid and telephony and SmsManager	30. Androids. Intent. Category. HOME
15. Landroid and content and pm and ApplicationInfo	31. Androids. Hardwares. screens.portraits
16.Androids .Permissions. Access_Cache_Filesystem	32. Ljava and util and TimerTask

The method start Service (), getPackageManager and openFileOutput() are commonly used by malware within this class. To start a programme without communicating with the users we used startService(); To get the information related to application we used getPackageManager(), no matter it is install or not; and we can used openFileOutput() to write data in a specific private record in the private folder of the application. In this sense, we represented the malware activity using the denotation $Lexical_2$, and we were able to extract a total of 32 features. Thus, L= $Lexical_1$+ $Lexical_2$ can be used to express the lexical vectors. Next, we looked up the matching embedding vector for lexical features identification using the word embedding. The features of lexical offer clear information gain for detecting Android malware, increasing the detection accuracy of the malware.

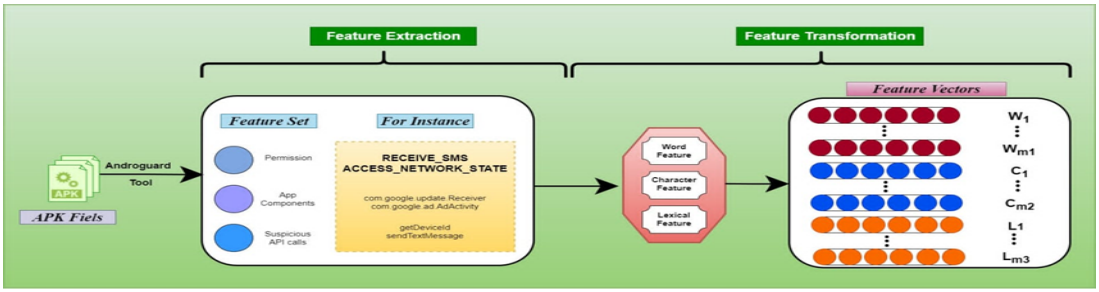

Figure 2. Overview of feature extraction & feature transformation

3.1.4. Feature Transformation

All behavioral information is encoded at the word, character, and lexical levels using graph convolutional networks. A set of applications is defined as $S = (x^i, y^i)$where N is the number of instances and x^i is an instance of the application.

In order to embed the graph's nodes, we extracted as many features into a vector space using the embedding approach. A collection of hidden variables represents the embedding, and every word is symbolized by a particular instance of these variables. Each malware sample is able to be presented by three different types of features based on multi-feature extraction technique: representation of word (represented by W), representation of character (represented as C), and lexical feature representation (represented by L). Then these three features can be fused together to create the combined features. The ultimate feature representation, represented as V = W+C+L is the progression of W, C, and L. In order to transfer the features from a dictionary of words, characters and lexical through a vector of real numbers in embedded vector space, the embeddings for every input example must be calculated. Following training, each word is associated with a distinct vector that appears as a column in weight matrix$E^{M\times K}$. This vector is K-dimensional, and in which K equals 200. The final lexical and character representations resemble word representations.

3.2. Proposed Methodology

Over the last few years, deep learning techniques such as hybrid models, convolution neural networks, deep neural networks, and recurrent neural networks (RNN) have performed a variety of critical roles in malware detection, which is recognized as an emerging research field. The widely used method to detect malware is convolution neural network (CCN) which uses image-based detection techniques.

After the training of CNN, it can obtain advanced features by learning the weight of train-able filters. However, the method of detecting malware through image-based techniques is now old, and graph-based methods are widely used to detect malware in Android. However, it is hard to apply convolution directly on a graph. So we presented a new technique that is based on an Effective Graph Convolutional Network (GCN) that directly works on graph-based malware. An exclusive adjacency matrix was created. Figure 3 shows the working of our proposed approach. Each node can have a certain number of ordered neighbors constructed around it. By using this adjacency matrix, we first represented regular grid data as a graph, and GCNs used to collect the characteristics of all nearby nodes for every node.

3.2.1. Deep Graph-based Convolutional Network

A variant of GCNs was presented for the purpose of simulating advanced graphical semantics. A set of tuples (A, X) is added, containing the fundamental graph structure's adjacency matrix A, and embedded matrix X., to facilitate efficient information transmission on the graph. We employ GCN to categorize the tuples (A, X) input into the categories of malware and benign. A graph is represented by G (Edg, Ver) with vertices set $\{Ver = \{Ver_1 \cdots Ver_n\}$ and edges set $Edg \in Ver \times Ver$.There is an embedded vector connected to every node. As was discussed in the previous section, all static features, including permissions and components, are converted into a collection of semantic features. The vectors that are embedded can be thought of as characteristics of each node in the network, which in our instance are knowledge of words, characters and lexical in a sentence. In reality, these nodes represent the APIs, permissions and components. Our objective is to identify the appropriate function∅ for the real-valued feature vector that the GCNs encode with pertinent information about its neighborhood.

By iteratively collecting vectors from its neighboring nodes, we can generate node vectors. An embedded matrix X with a size of $R^{M\times K}$defines all nodes embedded vectors in a network. The matrix of adjacency A suggests the structure of the local neighborhood. By learning a map that converts every graph into a vector space, we try to learn the graph's representation [51]. The learned vectors' geometric relationships in this space represent the structure of graph knowledge, which can be an input for other Deep learning activities [48] [52].

We created a special adjacency matrix A for building graphs. The neighboring nodes could be determined as candidates of the core node based on the relative positions of candidate nodes with regard to the middle node. This procedure may involve the use of certain node selection techniques. To acquire the ordering indexes, the adjacency matrix is subjected to an order. In order to create a defined number of candidates for every node in the graph, we selected the most above n nodes as the possible neighbor nodes of the middle node. Similar relative neighbors are determined for the nodes from several graphs. This technique can be used to create the adjacency matrix, which lists the connections (edges) between the nodes. Then, the adjacency matrix is recreated with the help of normalization operation. The process is explained as follows:

$$\widehat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$$

$$\tilde{A} = \hat{D}^{\frac{1}{2}} \hat{A} \hat{D}^{\frac{1}{2}}$$

The identity matrix I adds self-loop interactions, while the degree matrix of diagonal node D normalizes the generated A.

GCNs are primarily used to gather data from neighbors and understand how the community is represented. The inputs to the GCNs are the adjacency matrix A and the embedding matrix X. This graph structure framework uses induction to discover each node's embedding. It's important to note that neighborhood data is frequently combined via matrix multiplication $(\tilde{A} H^{k-1} W^{k})$. The method of aggregation differs from our own. In our example, the aggregated operation may be defined as follows:

$$\tilde{H}^{K} = ReLU(\sum_{u \in N(Ver)} \square (ZW^{k}))$$

$$Z = Agg\ (\tilde{A}, H^{k-1})$$

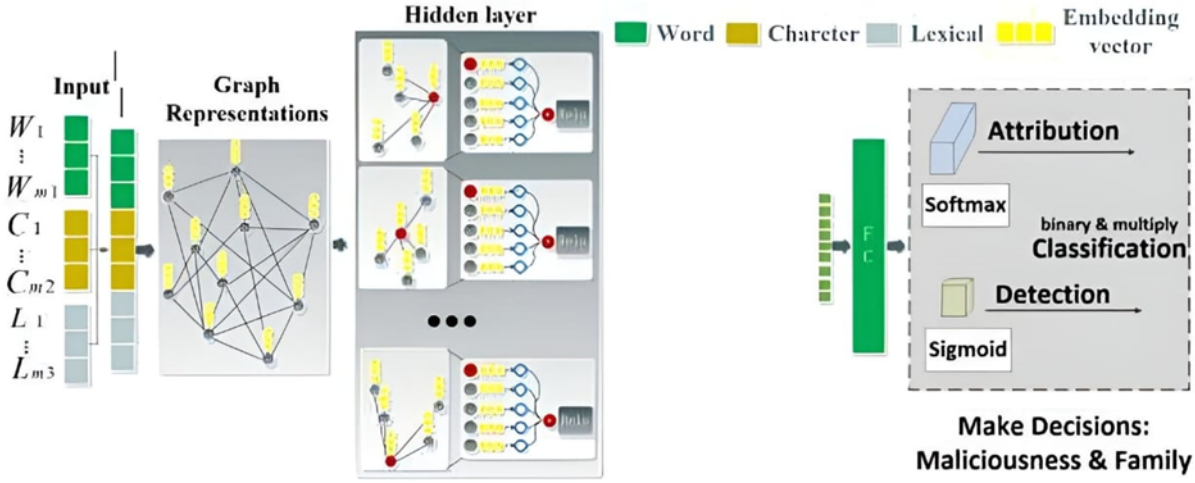

Figure 3. Flow diagram of proposed approach

4. Results and Discussion

4.1. Datasets

The AMD and Drebin databases are the sources of the datasets used in this work. The dataset used in this experiment are summarized in Table 4.

Table 4. Summary of Datasets

Datasets	Benign	Malware	Year
AMD	2100	24,553	2010-16
Drebin	123,453	5560	2010-12

The availability of representative data poses a significant challenge in Android virus detection studies. AMD[42] is the most recent dataset, containing 24,553 malware files gathered from 2010 - 2016. The samples of malware are divided into 135 variations across 71 malware families.

The Android malware dataset Drebin was gathered between Aug 2010 and Oct 2012. There are 5,560 malware Android apps in it, categorized into 179 families[43].

4.2. Experimental Setup

The experiment we conducted in a 64 bit Windows 10 pro OS with Inter (R) Core (TM) i3-4030U CPU @ 1.90 GHz with 16 GB RAM. The software used in this paper to run the code is Google Colab. The efficiency of malware detection is greatly influenced by the model parameters that are chosen. The efficiency of malware detection is greatly influenced by the model parameters that are selected. The hyper parameters setting of Effective GCN is described in Table 5.

After them, the number of convolution layers in the graph is represented by the GCN-layers.The GCN-filters show how many output channels there are in the Graph Convolution layer. Each node's number of neighbors is indicated by the GCN-neighbors. The batch amount is indicated by the batch size. The number of repetitions used for model training is shown by the epochs. Moreover, Dropout is a method for preventing overfitting that involves randomly eliminating nodes during training. We apply 50% dropout

at the GCN in our experiment. We experimented with these parameters and observed that little changes had no effect on the outcome.

Table 5. Hyper parameter setting

Number of Parameters	Total Values
GCN-layers	2
Embedding	200
Dropout	50%
Epochs	40
Batch-size	128
GCN-neighbors	25
GCN-filters	50

4.3. Evaluation Metrics

Evaluation metrics are measurable metrics that are used to evaluate a system's performance in the context of machine learning and data analysis. These metrics aid in assessing the model's performance about its ability to provide precise predictions or classifications. The particular job at hand and the type of data being examined will determine which assessment metrics are employed.

The quantity of malicious applications that are accurately categorized as malware is known as True Positives (TP). In contrast to False Positives (FP), which show how many benign apps are mistakenly labeled as malware. True Negatives (TN) show how many benign applications are accurately categorized as benign. In contrast, False Negatives (FN) indicates the quantity of malware samples that are mistakenly categorized as benign.

To prove the classifier's performance numerically, we use several different machine learning performance metrics:

Accuracy: The overall amount of samples which are accurately classified as benign or malicious is known as accuracy. It is denoted by (Acc).

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Precision: The proportion of accurately positively identified occurrences to all positive labeled instances is known as precision (P).

$$Precision = \frac{TP}{TP+FP}$$

Recall: A recall (R) is a proportion of all accurately labeled positive cases to all occurrences that should have had a positive label.

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

F-Score: The harmonic mean of Precision and Recall is the F-score (F).

$$F-score = 2\frac{Precision*Recall}{Precision+Recall}$$

The result of the experiment in this paper is a generalized version of CNN models for Android malware detection which is a GCN-based approach that detects malware on graph-based data. As compared with other approaches our proposed model gives accuracy is 99.69% and the other two datasets that are compared with our approach gave an accuracy of 99.24% and 99.19%.

In our proposed approach we used a GCN-based technique for malware detection with the help of a static approach. The static approach is used to detect malware by using code segments. Then these binary codes are run on python to read the datasets and convert them into vector format. Then we take the graph form of the input dataset and extract features. These extracted features train the GCN model for better results. GCN is compared with other deep learning models and we get better accuracy and better results as compared to other models.

4.4. Comparison with Different Algorithms

To verify reliability, a comparison was done using either a single model or a mix of those models. The CNN+ Bi-LSTM baseline, whose accuracy and PRF are comparable to our suggested methods, completely demonstrates the generalization capabilities of both CNN and Bi-LSTM. The three algorithms (CNN, LSTM, and Bi-LSTM) that were tested performed quite similarly overall; however, Bi-LSTM performed somewhat better than the other two methods [53] [54]. The rationale is that Bi-LSTM can record information

of context dependency in terms of both forward and future and the others cannot. Below Table 6 shows the experimental results of some different algorithm comparisons[44].

Table 6. Experimental results of different algorithms comparison

Models	DREBIN datasets				AMD datasets			
	Accuracy	**Precision**	**Recall**	**F1-score**	**Accuracy**	**Precision**	**Recall**	**F1-score**
Bi-LSTM	99.3	99.15	99.47	99.31	99.24	98.71	99.76	99.23
Cnn	99.24	99.04	99.47	99.26	98.95	98.82	99.05	98.93
LSTM	98.97	99.14	98.83	98.99	98.42	98.92	97.87	98.39
Cnn+Bi-LSTM	99.13	98.94	99.36	99.15	99.24	99.17	99.29	99.23
CNN+LSTM	98.92	99.46	98.41	98.93	98.84	98.93	98.7	98.81
GCN	99.69	99.54	99.82	99.7	99.47	99.52	99.4	99.46

Figure 4 shows the comparison between CNN and GCN. We compared this model with our suggested approach using evaluation metrics to evaluate the performance of our model. In terms of accuracy, precision, recall and f-measure the results of CNN with Drebin and AMD datasets are lower than our proposed approach. Using the Drebin dataset, the GCN model outperformed the CNN model, which achieved accuracy of 99.24%, precision of 99.04%, recall of 99.47%, and F-measure of 99.26%. The GCN model achieved greater accuracy was 99.69%, with precision was 99.54%, recall of 99.82%, and F-measure of 99.7%. Using the AMD dataset, the GCN model outperformed the CNN model, which scored 98.95% accuracy, 98.82% precision, 99.05% recall, and 98.93% F-measure, with a 99.47% accuracy, 99.52% recall, and 99.46% F-measure.

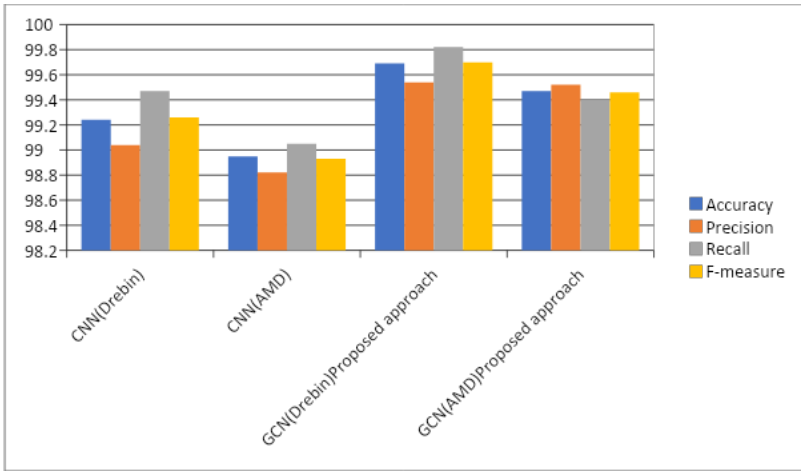

Figure 4. Comparison between CNN and GCN

Figure 5 shows the comparison between LSTM and GCN (proposed approach). We compared this model with our suggested approach using evaluation metrics to evaluate the performance of our model. In terms of accuracy, precision, recall and f-measure the results of LSTM with Drebin and AMD datasets are lower than our proposed approach. Using the Drebin dataset, the GCN model outperformed the LSTM approach, which achieved an accuracy of 98.97%, precision at 99.14%, recall of 98.83%, and F-measure of 98.99%. The GCN model achieved greater accuracy with 99.69%, with precision at 99.54%, recall of 99.82%, and F-measure of 99.7%. The GCN model outperformed the LSTM model, which obtained accuracy of 98.42%, precision with 98.92%, recall of 97.87%, and F-measure of 98.39%, using the AMD dataset, achieving 99.47% accuracy, 99.52% precision, 99.4% recall, and F-measure of 99.46%.

Figure 6 shows the comparison between Bi-LSTM and GCN (proposed approach). We compared this model with our suggested approach using evaluation metrics to evaluate the performance of our model. In terms of accuracy, precision, recall and f-measure the results of Bi-LSTM with Drebin and AMD datasets are lower than our proposed approach. Using the Drebin dataset, the GCN model outperformed the Bi-LSTM approach, which achieved accuracy of 99.3%, precision at 99.15%, recall of 99.47%, and F-measure

of 99.31%. The GCN model achieved greater accuracy at 99.69%, and precision with 99.54%, recall of 99.82%, and F-measure of 99.7%. Using the AMD dataset, the GCN model outperformed the Bi-LSTM model, which obtained accuracy of 99.24% with precision at 98.71%, recall of 99.76%, and F-measure of 99.23%, with an accuracy of 99.47%, precision at 99.52%, recall of 99.4%, and F-measure of 99.46%.

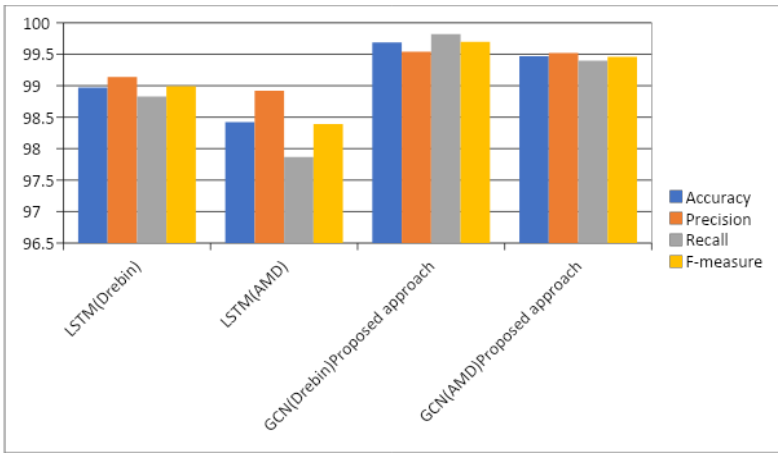

Figure 5. Comparison between LSTM and GCN

Figure 7 shows the comparison between CNN+LSTM and GCN (Proposed approach). We compared this model with our suggested approach using evaluation metrics to evaluate the performance of our model. In terms of accuracy, precision, recall and f-measure the results of CNN+LSTM with Drebin and AMD datasets are lower than our proposed approach. Using the Drebin dataset, the GCN model outperformed the CNN+LSTM approach, which obtained accuracy of 98.92%, precision with 99.46%, recall of 98.41%, and F-measure of 98.93%. The GCN model achieved greater accuracy at 99.69%, with precision with 99.54%, recall of 99.82%, and F-measure of 99.7%. Compared to the CNN+LSTM model, which obtained accuracy of 98.84%, precision with 98.93%, recall of 98.7%, and F-measure of 98.81%, the GCN model obtained accuracy with 99.47% with AMD dataset, with precision with 99.52%, recall of 99.4%, and F-measure of 99.46%.

Figure 8 shows the comparison between CNN+Bi-LSTM and GCN (Proposed approach). We compared this model with our suggested approach using evaluation metrics to evaluate the performance of our model. In terms of accuracy, precision, recall and f-measure the results of CNN+Bi-LSTM with Drebin and AMD datasets are lower than our proposed approach. Using the Drebin dataset, the GCN model outperformed the CNN+Bi-LSTM approach, which obtained accuracy of 99.13%, precision with 98.94%, recall of 99.36%, and F-measure of 99.15%.

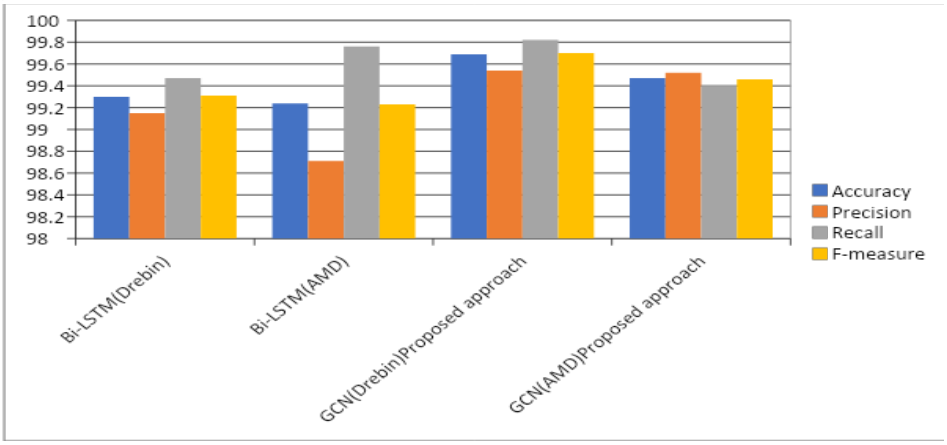

Figure 6. Comparison between Bi-LSTM and GCN

The GCN model achieved greater accuracy with 99.69%, and precision with 99.54%, recall of 99.82%, and F-measure of 99.7%. The GCN model outperformed the CNN+Bi-LSTM model, which obtained accuracy of 99.29%, precision with 99.17%, recall of 99.29%, and F-measure of 99.23%, with an AMD dataset accuracy of 99.47%, precision with 99.52%, recall of 99.4%, and F-measure of 99.46%.

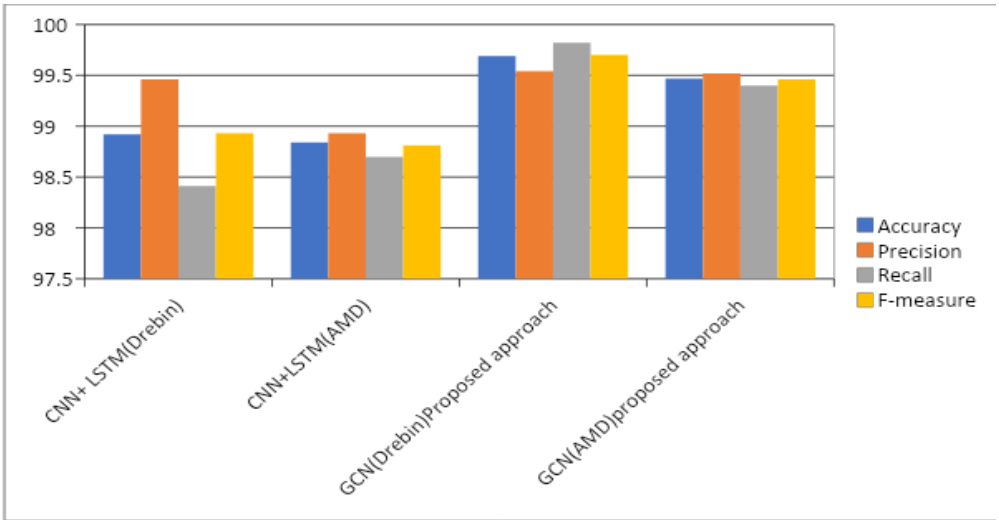

Figure 7. Comparison between CNN+LSTM with GCN

In 2018, Rui used Graph representation learning to detect malware. Training a graph classifier can be challenging due to the variety of input graph sizes and shapes available. They devised a batch-training approach to address this problem. They used to collect 5560 malware from DREBIN datasets and benign software from three different sources. The author did not offer benign datasets and used unequal positive or negative samples for testing and training, preventing precise comparison. However, this model of GCN is achieving 99% accuracy in detecting Android malware[25] [49]as shown in Table 6.

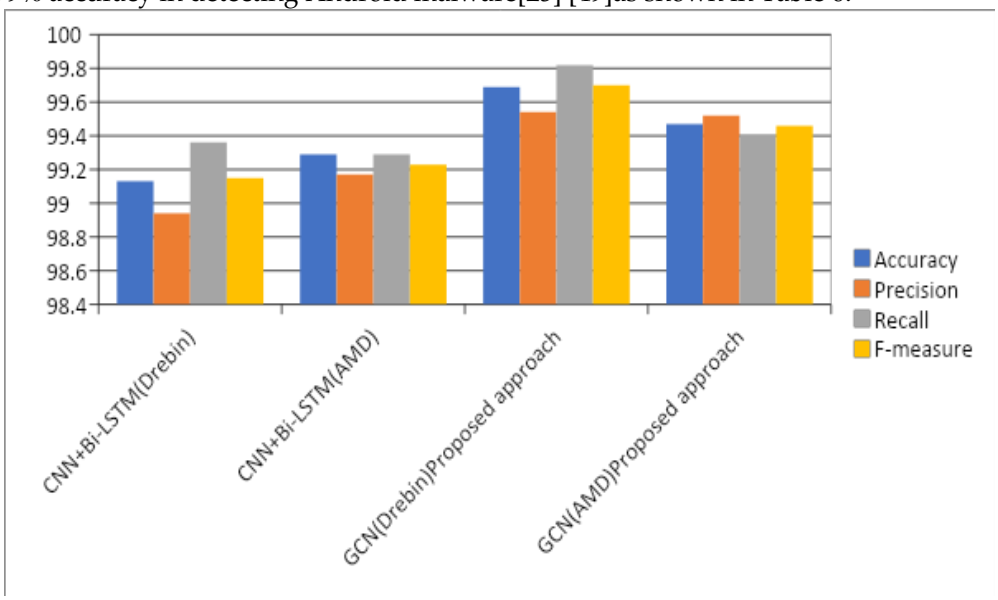

Figure 8. Comparison between CNN + Bi-LSTM with GCN

This result (GCNs) closely matches our method in terms of accuracy and PRF. GCNs provide deeper semantic information, but lack long-distance relationships between nodes, making them insufficient for describing complicated malware. Combining GCNs with LSTM and Bi-LSTM results in lower overall performance compared to other GCN-based models due to LSTM and Bi-LSTM's inability to utilize long sequence data.

Marcheggiani (2017) demonstrated the effectiveness of GCNs in Natural Language Processing and presented a GCN model to encoding the syntactic data at the level of word. The study found that stacking syntax-based GCNs on top of bidirectional Long Short-Term Memory (LSTM) layers enhances their modeling capabilities. The author implemented and used the technique (Bi-LSTM+GCNs) in the NLP task. The Bi-LSTM+GCNs outperform in terms of comparison with our technique on the basis of two comparable datasets[45] [46].

Table 7. Experimental results of different feature comparison

Number of Features	DREBIN datasets	AMD datasets

	Accuracy	Precision	Re-call	F1-score	Accuracy	Precision	Re-call	F1-score
Character	88.01	89.39	85.99	87.65	88.3	91.1	84.16	87.5
Words	98.92	99.24	98.61	98.93	98.95	99.87	97.98	98.92
Lexical	89.24	89.94	87.76	88.83	85.66	85.35	85.66	85.5
GCN	99.69	99.57	99.82	99.7	99.47	99.52	99.4	99.46

Table 7 indicates experimental results of different feature comparison with character, words and lexical with our proposed model [44] [47].

5. Conclusion

In this paper, we presented a GCN-based algorithm for detecting Android malware. It is platform-independent for both packed and unpacked malware. First, we allow the model to click multiple types of some semantic information. Then we proposed a Graph Convolutional Network (GCN) that works and operates directly on graphs. Then we use a static approach to extract the features. The static approach is used to detect malware by using code segments. Then these binary codes are run on python to read the datasets and convert them into vector format. Then we take the graph form of the input dataset and extract features. These extracted features train the GCN model for better results. Then these extracted features are passed through a GCN layer and the same classifier. The classifier obtained accuracy of 99.69% in separating benign from malware images. Then we compared our model with pre-trained CNN-based models and GCN-based models. We also perform dataset comparisons in combination with GCN models. The results show that our proposed methodology outperformed in detecting malware.

Reference

1. Bilot, T., et al., A Survey on Malware Detection with Graph Representation Learning. arXiv preprint arXiv:2303.16004, 2023.
2. Zhou, J., et al., Graph neural networks: A review of methods and applications. AI open, 2020. 1: p. 57-81.
3. Bhat, P., S. Behal, and K. Dutta, A system call-based android malware detection approach with homogeneous & heterogeneous ensemble machine learning. Computers & Security, 2023. 130: p. 103277.
4. Molina-Coronado, B., et al., Towards a fair comparison and realistic evaluation framework of android malware detectors based on static analysis and machine learning. Computers & Security, 2023. 124: p. 102996.
5. Ding, L., X. Chen, and Y. Xiang, Negative-supervised capsule graph neural network for few-shot text classification. Journal of Intelligent & Fuzzy Systems, 2021. 41(6): p. 6875-6887.
6. Stamp, M., M. Alazab, and A. Shalaginov, Malware analysis using artificial intelligence and deep learning. Vol. 1. 2021: Springer.
7. Kawoosa, A.I., et al., Using machine learning ensemble method for detection of energy theft in smart meters. IET Generation, Transmission & Distribution, 2023. 17(21): p. 4794-4809.
8. Ventures, C., Cybercrime Damages $6 trillion by 2021. Cybersecurity Ventures, 2019.
9. Yadav, P., et al., EfficientNet convolutional neural networks-based Android malware detection. Computers & Security, 2022. 115: p. 102622.
10. Kural, O.E., E. Kiliç, and C. Aksaç, Apk2Audio4AndMal: Audio Based Malware Family Detection Framework. IEEE Access, 2023. 11: p. 27527-27535.
11. Nath, H.V. and B.M. Mehtre. Static malware analysis using machine learning methods. in Recent Trends in Computer Networks and Distributed Systems Security: Second International Conference, SNDS 2014, Trivandrum, India, March 13-14, 2014, Proceedings 2. 2014. Springer.
12. Raza, A., et al., TL-GNN: Android Malware Detection Using Transfer Learning. Applied AI Letters, 2023: p. e94.
13. Ojugo, A. and A. Eboka, Signature-based malware detection using approximate Boyer Moore string matching algorithm. International Journal of Mathematical Sciences and Computing, 2019. 5(3): p. 49-62.
14. Zhang, X., et al., MalCaps: a capsule network based model for the malware classification. Processes, 2021. 9(6): p. 929.
15. Abid, M.K., et al., IoT Environment Security and Privacy for Smart Homes. Journal of Information Communication Technologies and Robotic Applications, 2022. 13(1): p. 15-22.
16. Jung, J., et al. Android malware detection using convolutional neural networks and data section images. in Proceedings of the 2018 conference on research in adaptive and convergent systems. 2018.
17. Lachtar, N., D. Ibdah, and A. Bacha, Toward mobile malware detection through convolutional neural networks. IEEE Embedded Systems Letters, 2020. 13(3): p. 134-137.
18. Kim, T., et al., A multimodal deep learning method for android malware detection using various features. IEEE Transactions on Information Forensics and Security, 2018. 14(3): p. 773-788.
19. Yen, Y.-S. and H.-M. Sun, An Android mutation malware detection based on deep learning using visualization of importance from codes. Microelectronics Reliability, 2019. 93: p. 109-114.
20. Zhang, H., et al., A multiclass detection system for android malicious apps based on color image features. Wireless Communications and Mobile Computing, 2020. 2020: p. 1-21.
21. Abid, M.K., Z.U.R. Zia, and S. Farid, Security and Privacy for Future Healthcare IoT. Journal of Computing & Biomedical Informatics, 2022. 4(01): p. 132-140.
22. de Oliveira, A. and R.J. Sassi, Chimera: an android malware detection method based on multimodal deep learning and hybrid analysis. TechRxiv, 2020.
23. Nasir, H., et al., Cloud Computing Security via Intelligent Intrusion Detection Mechanisms. International Journal of Information Systems and Computer Technologies, 2024. 3(1): p. 84-92.
24. Zhong, P., D. Wang, and C. Miao, EEG-based emotion recognition using regularized graph neural networks. IEEE Transactions on Affective Computing, 2020. 13(3): p. 1290-1301.
25. Zhu, R., et al., Android malware detection using large-scale network representation learning. arXiv preprint arXiv:1806.04847, 2018.
26. Wang, W., et al. Malware traffic classification using convolutional neural network for representation learning. in 2017 International conference on information networking (ICOIN). 2017. IEEE.
27. Chen, H., et al. Android malware classification using XGBoost based on images patterns. in 2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC). 2018. IEEE.

28. Darus, F.M., N.A. Ahmad, and A.F.M. Ariffin. Android malware classification using XGBoost on data image pattern. in 2019 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS). 2019. IEEE.
29. Huang, W., et al. MixDroid: a multi-features and multi-classifiers bagging system for Android malware detection. in AIP conference proceedings. 2018. AIP Publishing LLC.
30. Tahir, U., et al., Enhancing IoT Security through Machine Learning-Driven Anomaly Detection. VFAST Transactions on Software Engineering, 2024. 12(2): p. 01-13.
31. Gu, S., S. Cheng, and W. Zhang. From image to code: executable adversarial examples of android applications. in Proceedings of the 2020 6th international conference on computing and artificial intelligence. 2020.
32. Ding, Y., et al., Single-polarization photonic crystal fiber filter composed of elliptical gold films. Optical Engineering, 2020. 59(7): p. 076103-076103.
33. Lekssays, A., B. Falah, and S. Abufardeh. A Novel Approach for Android Malware Detection and Classification using Convolutional Neural Networks. in ICSOFT. 2020.
34. Yuan, Z., Y. Lu, and Y. Xue, Droiddetector: android malware characterization and detection using deep learning. Tsinghua Science and Technology, 2016. 21(1): p. 114-123.
35. Karbab, E.B., et al. Dysign: dynamic fingerprinting for the automatic detection of android malware. in 2016 11th International Conference on Malicious and Unwanted Software (MALWARE). 2016. IEEE.
36. Ramzan, M., et al., A review study on smart homes present challenges concerning awareness of security mechanism for Internet of Things (IoT). Journal of Computing & Biomedical Informatics, 2024.
37. John, T.S., T. Thomas, and S. Emmanuel. Graph convolutional networks for android malware detection with system call graphs. in 2020 Third ISEA Conference on Security and Privacy (ISEA-ISAP). 2020. IEEE.
38. Gao, H., S. Cheng, and W. Zhang, GDroid: Android malware detection and classification with graph convolutional network. Computers & Security, 2021. 106: p. 102264.
39. Yumlembam, R., et al., Iot-based android malware detection using graph neural network with adversarial defense. IEEE Internet of Things Journal, 2022.
40. Li, S., et al., Intelligent malware detection based on graph convolutional network. The Journal of Supercomputing, 2022. 78(3): p. 4182-4198.
41. Wu, Y., et al., DeepCatra: Learning flow-and graph-based behaviours for Android malware detection. IET Information Security, 2023. 17(1): p. 118-130.
42. Wei, F., et al. Deep ground truth analysis of current android malware. in Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14. 2017. Springer.
43. Arp, D., et al. Drebin: Effective and explainable detection of android malware in your pocket. in Ndss. 2014.
44. Pei, X., L. Yu, and S. Tian, AMalNet: A deep learning framework based on graph convolutional networks for malware detection. Computers & Security, 2020. 93: p. 101792.
45. Marcheggiani, D. and I. Titov, Encoding sentences with graph convolutional networks for semantic role labeling. arXiv preprint arXiv:1703.04826, 2017.
46. Ahmed, F., Sumra, I. A., & Jamil, U. (2024). A Comprehensive Review on DDoS Attack in Software-Defined Network (SDN): Problems and Possible Solutions. Journal of Computing & Biomedical Informatics, 7(01).
47. Batool, S., Abid, M. K., Salahuddin, M. A., Aziz, Y., Naeem, A., & Aslam, N. (2024). Integrating IoT and Machine Learning to Provide Intelligent Security in Smart Homes. Journal of Computing & Biomedical Informatics, 7(01), 224-238.
48. Abbas, M., Arslan, M., Bhatty, R. A., Yousaf, F., Khan, A. A., & Rafay, A. (2024). Enhanced Skin Disease Diagnosis through Convolutional Neural Networks and Data Augmentation Techniques. Journal of Computing & Biomedical Informatics, 7(01).
49. Hussain, S.K., Ramay, S.A., Shaheer, H., Abbas T., Mushtaq M.A., Paracha, S., & Saeed, N. (2024). Automated Classification of Ophthalmic Disorders Using Color Fundus Images, Volume: 12, No: 4, pp. 1344-1348 DOI:10.53555/ks.v12i4.3153
50. Aqsa Ijaz, Ammar Ahmad Khan, Muhammad Arslan, Ashir Tanzil, Alina Javed, Muhammad Asad Ullah Khalid, & Shouzab Khan. (2024). Innovative Machine Learning Techniques for Malware Detection. Journal of Computing & Biomedical Informatics, 7(01), 403–424.
51. Mahmood, M. A., Malik, H., Khan, A. H., Adnan, M., & Khan, M. I. A. (2022). Neural Network-Based Prediction of Potential Ribonucleic Acid Aptamers to Target Protein. Journal of Computing & Biomedical Informatics, 4(01), 21-36.

52. Khan, A. H., Malik, H., Khalil, W., Hussain, S. K., Anees, T., & Hussain, M. (2023). Spatial Correlation Module for Classification of Multi-Label Ocular Diseases Using Color Fundus Images. Computers, Materials & Continua, 76(1).
53. Abbas, F., Iftikhar, A., Riaz, A., Humayon, M., & Khan, M. F. (2024). Use of Big Data in IoT-Enabled Robotics Manufacturing for Process Optimization. Journal of Computing & Biomedical Informatics, 7(01), 239-248.
54. Shah, A. M., Aljubayri, M., Khan, M. F., Alqahtani, J., Sulaiman, A., & Shaikh, A. (2023). ILSM: Incorporated Lightweight Security Model for Improving QOS in WSN. Computer Systems Science & Engineering, 46(2).