ENHANCING GRAPH CONVOLUTIONAL NETWORK WITH LABEL PROPAGATION AND RESIDUAL FOR MALWARE DETECTION

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Cyber Security

by

ARAVINDA SAI GUNDUBOGULA
B.Tech., Gayatri Vidya Parishad College of Engineering Autonomous, 2018

2023
Wright State University

Wright State University
COLLEGE OF GRADUATE PROGRAMS AND HONOR STUDIES

April 27, 2023

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Aravinda Sai Gundubogula ENTITLED Enhancing Graph Convolutional Network with Label Propagation and Residual for Malware Detection BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Cyber Security.

Dr. Lingwei Chen, Ph.D.
Thesis Director

Thomas Wischgoll, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on Final Examination

Dr. Lingwei Chen, Ph.D.

Dr. Meilin Liu, Ph.D.

Dr. Junjie Zhang, Ph.D.

Dr. Shu Schiller, Ph.D.
Interim Dean, College of Graduate Programs & Honors Studies

ABSTRACT

Gundubogula, Aravinda Sai. M.S.C.S., Department of Computer Science and Engineering, Wright State University, 2023. *Enhancing Graph Convolutional Network with Label Propagation and Residual for Malware Detection.*

Malware detection is a critical task in ensuring the security of computer systems. Due to a surge in malware and the malware program sophistication, machine learning methods have been developed to perform such a task with great success. To further learn structural semantics, Graph Neural Networks abbreviated as GNNs have emerged as a recent practice for malware detection by modeling the relationships between various components of a program as a graph, which deliver promising detection performance improvement. However, this line of research attends to individual programs while overlooking program interactions; also, these GNNs tend to perform feature aggregation from neighbors without considering any label information, and significantly suffer from over-smoothing on node presentations.

To address these issues, this thesis constructs a graph over program collection to capture the program relations, and designs two enhanced graph convolutional network (GCN) architectures for malware detection. More specifically, the first proposed GCN model incorporates label propagation into GCN to take advantage of label information for facilitating neighborhood aggregation, which is used to propagate labels from the labeled nodes to the unlabeled nodes; the second proposed GCN model introduces residual connections between the original node features and the node representations produced by GCN layer to enhance the flow of information through the network and address over-smoothing issue. The experimental results after substantial experiments show that the proposed models outperform the baseline GCN and classic machine learning methods for malware detection, which confirm their effectiveness in program representation learning using either label propagation or residual connections and malware detection using program graph. Furthermore, these models can be used for other graph-based tasks other than malware detection, demonstrating their versatility and promise.

Contents

List of Figures

List of Tables

Acknowledgment

I want to express my gratitude to Dr. Lingwei Chen, who served as my thesis advisor, for his direction, encouragement, and support during the whole process. His skills and commitment to my academic progress have been critical in defining my research and assisting me in reaching this stage. I also want to express my gratitude to my committee members, Dr. Meilin Liu and Dr. Junjie Zhang, for taking the time to review my thesis; your conversation, ideas, and feedback have been really helpful. Finally, I want to convey my gratitude to my family and loved ones for their constant support and encouragement. Their love and faith in me have carried me through the highs and lows of this journey, allowing me to reach this stage. Thank you all once more for your invaluable assistance.

Introduction

Malware has become a far bigger hazard in recent years, putting both people and businesses at serious risk. Traditional malware detection techniques based on signatures and heuristics are losing effectiveness as malware volume and complexity increase. The utilization of machine learning methods in malware detection has garnered significant attention as a major mean to address this issue. To capture the intricate relationships between various program components, graph neural networks (GNNs) have become a potential neural architecture for detecting malware.

In this thesis, we propose to analyze and predict malware or benign files using various graph convolutional network (GCN) (i.e., one of the most popular and significant GNNs) iterations over program interactions instead of individual program structures. The overall goal is to show how GCNs can be used to detect malware in a more effective and efficient way and emphasize their advantages over more conventional techniques. By doing this, we want to stimulate additional study in this field and advance ongoing efforts to enhance cybersecurity. In this chapter, we first give some basic information on malware detection, including both conventional techniques and more recent developments in machine learning. Next, we go over the reasons we wanted to do this research, the drawbacks of current methods, and the possible advantages of using GCNs. Then, we present our contributions to the field of malware detection, including the creation of GCN-based models and the assessment of their effectiveness using a real-world dataset. Finally, we describe the thesis organization and the subsequent chapter structure.

1.1 Background

Malware, an extremely popular term making rounds in all the news, is a kind of software that has malicious intentions. Malware has caused humongous losses to organizations. According to SonicWall's 2022 Cyber Threat Report [53], malware attacks are now sitting at 10.4 million per year; there are various types of malware, such as ransomware, adware, worms, PUPs and many more. When the threat caused by malware is huge, the measures to block and prevent attacks will be in place for any organizations. Malware detection is one of these preventive measures, which we are discussing throughout this paper.

Malware detection is a thorough examination of an application' s content to get to a conclusion whether or not it has malicious intents. This procedure usually entails the use of specialized software tools to scan the program' s code and behavior for signs of malicious activity. Various techniques are used during the malware detection process to investigate the program' s characteristics. Static analysis [46], which involves inspecting the program' s code without running it, and dynamic analysis, which involves running the program in a controlled environment to observe its behavior.

Malware detection techniques are broadly classified into two types: one is signature-based detection and the other one is behavior-based detection. Signature-based detection is a technique for detecting malware that searches the program' s code for specific patterns or signatures that are known to be associated with malware. The technique of behavior-based detection is used to identify malware by analyzing its behavior and actions on a system. But both the techniques have disadvantages that make them not useful on the account of new malware attacks. The next-generation adversaries who employ advanced methods of hashes, obfuscation techniques, and intelligent malware components are invulnerable to anti-malware signature and behavioral-based defenses [44, 55]. It is no longer sufficient to detect and respond.

Machine learning provides the predictive capability that can provide organizations with a much-needed advantage over their more advanced and malicious competitors. There are many machine learning algorithms that are used for malware detection. These algorithms are capable of analyzing large data-sets and identifying patterns that indicate malware. But these algorithms rarely consider the relations between the program files. Not leveraging such useful relations in a program data-set for malware detection is like throwing away banana and keeping the peel, and one does not get full pledged understanding of the data-set.

1.2 Motivation

We come to the questions: why analysing relations between program files, why not just use basic techniques of machine learning algorithms like others in this line of research, and what is so different about malware data-sets. Because malware authors frequently use complex techniques to obfuscate the malicious code and avoid detection by security software, analyzing relationships among malware files is an important aspect of malware detection. Malware can be designed to exploit system vulnerabilities, steal sensitive information, or disrupt a system's normal operation. Malware frequently interacts with other files, processes, or system components on a targeted system to achieve these goals.

Graph Neural Networks are a type of deep learning algorithm that can be used to analyze the relationships between various elements in a graph or network. Therefore, GNNs can be used to analyze the relationships between different files, processes, or system components that are targeted by malware in the context of malware detection, which can uncover patterns of activity that are suggestive of malware by examining these correlations. For instance, if a malware file is seen interacting in an unusual or suspicious way with a certain system component, this could be a sign that the file is harmful.

As such, finding new or undiscovered threats can also be assisted by analyzing relationships among malware files. To disguise their code and avoid being discovered by signature-based or behavior-based detection methods, malware developers frequently employ obfuscation techniques. However, GNNs can find patterns that are missed by conventional detection techniques by examining relationships in malware files. The accuracy and efficiency of malware detection systems can also be increased by looking at file relationships. Security analysts can identify threats more quickly and precisely by integrating GNNs into current malware detection systems and observing interactive behavioral patterns that are suggestive of malware. This increases the precision and potency of their malware detection tools.

Even with these many advantages of using GNNs for malware detection, there are still some hiccups for this line of research or models. The current malware detection works mainly use GNNs to study the connections between various parts in a program, application or a file, but overlook the more useful interactions among these programs, applications, and files. In addition, there GNNs only propagate features from neighbors and barely absorb the information from labels. What if we were able to propagate the labels as well? Another issue with using just basic GNNs is over-smoothing on the node representations. Over-smoothing occurs when the GNNs aggregate too much information from the neighboring nodes, resulting in a loss of discriminative power in the learned features. It would be a waste to use methods to improve GNNs and lose some of its powers. All these disadvantages motivate this thesis. In the next section, we briefly describe the contributions of this thesis.

1.3 Contributions

To address the aforementioned issues, we proposed to construct a graph over files regarding malware and benign files. As no existing relationships can be derived from the data, we calculated the distances between different file pairs and uses different filtering ideas (i.e., threshold filtering and top k neighbors) to prevent the graph from being over-dense and noisy, and better benefit the malware detection. Based on the constructed file graph, we further designed some variants of graph convolutional networks (GCNs) [31] (GCNs is one of the most popular and significant GNNs) to perform graph learning and malware detection. To propagate with labels, we need the help of label propagation, which is a semi-supervised machine learning algorithm that propagates labels to a large set of unlabeled data from a small set of labeled data. This is particularly helpful in the context of malware detection because it can be difficult to obtain a large set of labeled data for training machine learning models [41]. To deal with over-smoothing problem, we need the help of new method that introduces a new aggregation function with residual that incorporates information from lower-order and higher-order neighborhoods in the graph [9]. To this end, to facilitate GCNs to extract characteristics from the graph and classify the malware and benign files, we improved the GCN's performance by using two alternative techniques: GCN+LPA and GCN+Residual.

We evaluated the performance of the proposed approaches with a large-scale real-world dataset to four baseline methods: SVM, Random Forest, Neural Network, and Naive Bayes. We ran multiple trials with various combinations of training and testing datasets and different parameter settings. We measured the performance of the proposed methodology and baseline methods using accuracy. We discovered that our proposed variants of GCNs, including GCN+LPA and GCN+Residual surpassed all other methods for malware detection in terms of accuracy, which demonstrate their effectiveness in program representation learning using either label propagation or residual connections and malware detection using program graph.

1.4 Organization

The thesis is organized into six main chapters. Chapter 1, Introduction, provides an overview of the research problem and objectives of the study. Chapter 2, Literature Review, reviews the existing literature related to the malware detection, Graph Convolutional Networks and its enhanced methods and establishes the theoretical background of the study. Chapter 3, Malware detection using Enhancements of GCNs, describes the methodology used in the study, specifically the application of graph convolutional networks for malware detection. This chapter discusses about the disadvantages of GCN and enhancements added to GCN to improve it surpassing its disadvantages. Chapter 4, Experimental Results and Analysis, presents the results of the experimental study, including model evaluations, data analysis, and findings. Chapter 5, Challenges, discusses the limitations and challenges encountered during the study. Chapter 6, Future Work, outlines potential directions for future research based on the findings of the study. Chapter 7, Conclusion, concludes this thesis which summarizes the key contributions of the study and offers recommendations for practical applications of the research findings.

Literature Review

This chapter explains in detail the existing work related to this thesis and investigates various traditional and modern malware detection techniques. We will also discuss the benefits and drawbacks of these techniques, as well as recent advancements in the field, which will serve as motivations to propose our graph-based malware detection methods using GCN and its variants.

2.1 Traditional Methods

Traditional malware detection techniques include signature-based detection, behavior-based detection, heuristic-based detection and anomaly-based detection. Below are detailed studies for each technique, all of which focused on malware detection.

2.1.1 Signature-based Malware Detection

One of the more established techniques for detecting malware involves looking for recognizable patterns, or signatures, in files to determine if they are dangerous or not. Many anti-virus software packages are based on this strategy, which has been broadly adopted in the world of cyber security.

Jiang et al. (2007) suggested using structural data to generate automated signatures for polymorphic worms. Their process comprised taking the worm' s structural data and using it to build a Deterministic Finite Automaton (DFA) model, which was used to create signatures [44]. Ojugo et al. (2019) offers a rough Boyer Moore string matching algorithm for malware detection using signatures. By lowering the temporal complexity of signature matching, the method seeks to increase the effectiveness of the detection process. The authors test their strategy on a malware dataset and show that it produces high detection rates and few false positives [47]. Abbas et al. (2017) offers an device malware detection strategy for resource constrained devices that is modest in complexity and based on signatures. The authors present a lightweight signature matching method that eases the computational load on IoT devices with limited resources. The authors demonstrate that their method achieves excellent detection rates while keeping low cost by evaluating it against a collection of IoT malware [2]. The major disadvantages in using traditional signature based methods for malware detection are:

- They depend on an existing signature database to identify malware. As a result, these techniques may be unable to detect new and emerging threats and can only detect known malware [49].

- The quality of the signature database and the capacity to recognize polymorphic and obfuscated malware can both have an impact on how effective signature-based detection can be [58].

- Detection using signatures is susceptible to evasion strategies. Attackers can alter their malware and avoid detection using a number of methods, including code obfuscation, packing, and encryption. Because of this, it may be challenging for signature-based detection techniques to effectively identify malware, leading to a higher proportion of false negatives [6].

2.1.2 Behavior-based Malware Detection

Behavior-based malware detection is also known as behavior analysis. Behavior-based malware detection is a technique used to identify and stop malware by analyzing its behavior. This approach is particularly effective against new and unknown types of malware that have not yet been added to antivirus databases. Instead of relying on traditional signature-based detection methods, behavior-based detection focuses on the actions a piece of software takes on a system to identify potential threats.

The V. Chandola et al. (2009) study suggests a data mining-based approach for behavior-based malware identification. The authors use clustering and classification techniques to analyze network traffic data and show that a wide range of malware families may be detected with high accuracy [8]. In the study by J. W. Stokes et al. (2013) hierarchical hidden Markov models are proposed as a behavior-based detection method (HHMMs). The authors exhibit increased detection rates in comparison to conventional signature-based techniques by utilizing a multi-layered architecture of HHMMs to record various degrees of malware behavior [55]. S. Saxe et al. (2015) proposes recurrent neural networks to help behaviour based methods for malware detection (RNNs). The authors exhibit good detection rates for both known and unknown malware by training an RNN model on a sizable dataset of malware. However, this method also comes with many limitations such as false positives that will mark legitimate software as malware which affects the mission-critical processes of organizations. This method is very complex in nature, resource demanding, and cannot detect new versions of malware that has different behavior than the ones in the database [48]. However this traditional based technique has the following drawbacks:

- One drawback of behavior-based techniques is that they frequently rely on a set of predetermined rules or models, which may not be able to recognize novel and undiscovered malware that hasn't been seen before [54].

- The fact that behavior-based techniques might produce a lot of false positives due to the fact that both genuine software and malware can display identical actions is another drawback.

- Furthermore, because they frequently rely on manual analysis and interpretation of system data, behavior-based approaches may need considerable knowledge and work to design and maintain [33].

2.1.3 Heuristic-based Malware Detection

Heuristic detection entails employing a set of rules or heuristics to identify patterns or behaviors indicative of malware. These rules are frequently based on known malware characteristics, such as file size, file type, or specific code strings. Although heuristic-based detection can be effective in detecting known malware, it may be incapable of detecting new or unknown types of malware that do not match the established patterns.

Han et al. (2017) proposed a heuristic-based approach for detecting malware in embedded systems in their paper ”A Heuristic Approach to Malware Detection in Embedded Systems”. Their method detects malware based on code characteristics by combining signature-based and behavior-based heuristics. The authors show how their approach works on a real-world dataset of embedded system malware [25]. In their paper, Lee et al. (2017) proposed a heuristic-based malware detection framework for Android. To detect malware in Android applications, they use a set of heuristics based on API call sequences. On a dataset of real-world Android malware, the authors demonstrate the effectiveness of their approach [36]. In ”An Efficient Heuristic-Based Malware Detection System for Windows Executable”, Yasin et al. (2018) proposed an efficient heuristic-based malware detection system for windows executable. Their method detects malware using a combination of file header analysis and dynamic analysis based on a set of predefined heuristics [62]. The drawbacks of heuristic approaches include but not limited to:

- Heuristic-based methods rely on identifying specific patterns or behaviors that are indicative of malware. This means they may miss new or unknown malware that does not exhibit these patterns or behaviors.

- High False Positive Rate: Because legitimate software can exhibit similar patterns or behaviors to malware, heuristic-based methods can produce a high number of false positives.
- Updating and Maintenance: Heuristic-based methods require constant maintenance and updating to keep up with the evolving malware landscape. This can be a time-consuming and expensive procedure.
- Limited scalability is seen because heuristic-based methods rely on a set of predefined rules or signatures, they may not be scalable to large datasets or complex malware variants.

2.1.4 Anomaly-based Malware Detection

Anomaly-based detection entails examining system activity and spotting behavior that deviates from expected patterns. This method can be successful in identifying malware that was previously unknown since it is predicated on the idea that malware behavior will differ considerably from that of genuine software. Anomaly-based detection, however, can potentially produce false positives if legal software displays peculiar or unexpected behavior.

A behavior-based method to anomaly detection that keeps an eye on system behavior in order to spot unusual activity was put forth by Goldstein et al. (2000). Their method uses a set of behavioral rules to represent typical system behavior then compares observed system activity to this model to find anomalies. The authors tested their method against existing signature-based intrusion detection techniques and discovered that it outperformed them on a real-world intrusion detection dataset [23]. A set of rules are used in the variable-length anomaly signature (VLAS) method that Denning et al. (2009) proposed to identify unusual behavior in network data. The VLAS method creates signatures based on the distinctions between regular and irregular network traffic, then employs these signatures to find other instances of irregular behavior [20]. A thorough analysis of the detection of abnormal system calls with machine learning was presented by Lippmann et al. (2003). They identified anomalous system calls in system call traces by combining feature extraction and machine learning approaches. The authors tested their method on a real-world dataset and discovered that it outperformed other methods for anomaly identification [39].

When we observe how the above have drawbacks, there should be a solution better than those approaches to solve their disadvantages. That is when machine learning came into the picture. Machine learning offers many advantages as compared to traditional methods. Major advantage of all is the ability to predict unseen and new malicious software. Its another advantage that the machine learning model can train itself in finding new malicious software. This also makes it cost effective.

2.2 Machine Learning Methods

Machine learning techniques for malware detection have grown in popularity in recent years due to their ability to detect new and previously unknown malware. Machine learning algorithms can learn the characteristics of malware and use them to differentiate between malicious and benign software. Furthermore, machine learning algorithms can constantly adapt to new types of malware, increasing their effectiveness in detecting previously unknown threats.

Y. Zhou et al. (2012) in their paper proposed a technique to identify Android malware using machine learning, more specifically a Support Vector Machine (SVM) classifier. The paper evaluates the effectiveness of the proposed approach using real-world Android malware samples and demonstrates that the SVM classifier detects Android malware with high accuracy [68]. M. A. AlZu'bi et al. (2019) provided a comprehensive survey of recent re-

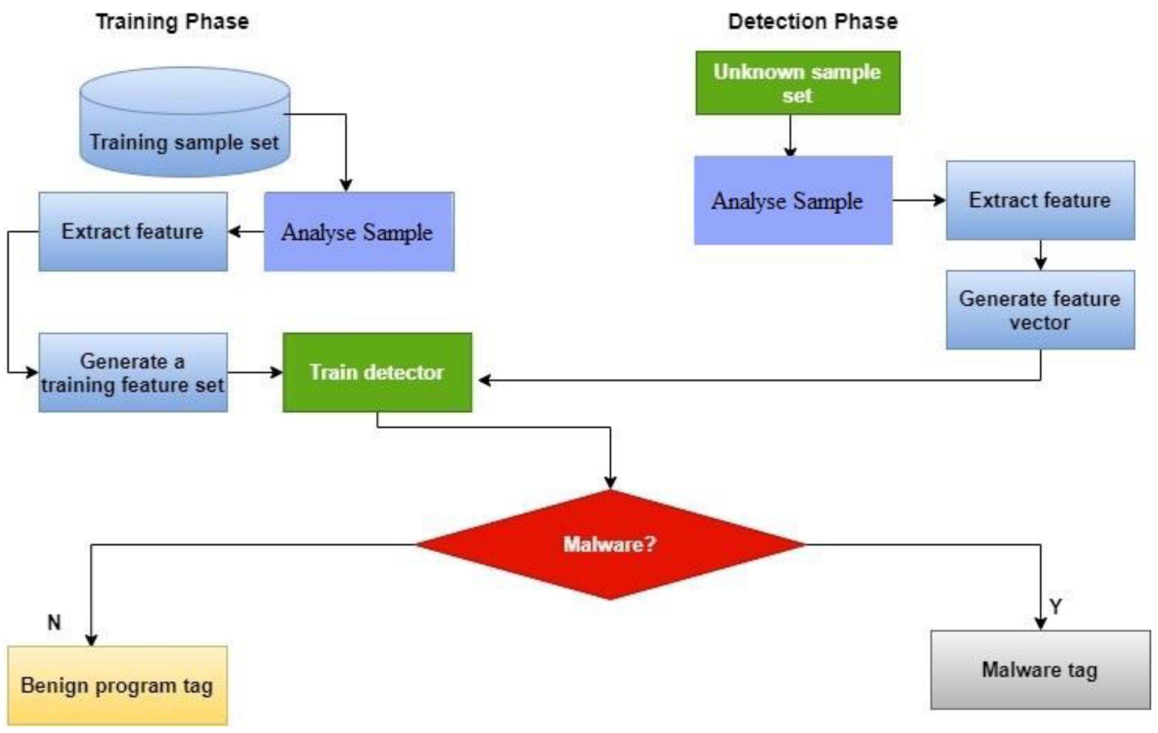

Figure 2.1: Flowchart on malware detection and analysis using machine learning [3]

search on malware detection using machine learning techniques in their survey paper. The paper examines the performance of various machine learning algorithms, such as random forests, neural networks, and deep learning techniques, in detecting malware [5]. There are several methods that can be used under the umbrella of machine learning based malware detection, which includes supervised learning, unsupervised learning, feature extraction and deep learning.

2.2.1 Supervised Learning

In malware detection, supervised learning is a machine learning technique that uses labeled data to train a model that can determine whether a sample is benign or malware. The labeled data is a collection of pre-classified samples from which the model learns the characteristics that distinguish malware from benign software. The model is trained on these samples and then used to predict whether new, unlabeled samples are malware or benign. Decision trees, support vector machines, and neural networks are some of the supervised learning algorithms that are commonly used in malware detection.

R. Vinayakumar et al. (2020) provided an introduction and review of the use of machine learning techniques for malware detection, which delves into various aspects of malware detection, such as feature selection, feature extraction, and classification using various machine learning algorithms [57]. S. Wang et al. (2015) proposed a malware detection framework based on supervised learning that employs feature selection techniques as well as four different classification algorithms: naive Bayes (NB), support vector machine (SVM), random forest (RF), and decision tree (DT). On a dataset of Windows executable files, the proposed framework was tested and compared to several other malware detection methods, including signature-based detection and unsupervised learning-based methods [59]. S. Haq et al. (2020) studied compared different feature selection methods for supervised learning-based malware detection. On a dataset of Windows executable files, the authors tested six different feature selection techniques and compared the performance of three different classification algorithms: NB, SVM, and RF. According to the data, a hybrid feature selection strategy that uses mutual information combining it with principle component analysis (PCA) fared the best in terms of accuracy and false positive rate [26].

2.2.2 Unsupervised Learning

In malware detection, unsupervised learning is a machine learning technique in which the algorithm is trained on a dataset with no labeled data. Instead, the algorithm is given the task of discovering patterns and similarities in the data on its own. This is useful when labeled data is either unavailable or prohibitively expensive to obtain. The unsupervised learning algorithm can cluster data and detect anomalies or outliers that could indicate the presence of malware. The following papers show some of the vast work done on unsupervised learning methods on malware detection.

R. Islam et al. (2019) proposed a novel unsupervised approach to malware traffic detection dependent on a feature extraction technique, followed by K-means clustering and one-class support vector machine [28]. H. Liu et al. (2017) proposed an unsupervised malware detection approach based on behavioral characteristics in their work. The proposed method grouped similar behaviors using clustering analysis and then built behavior models for each cluster [40]. P. F. Sequeira et al. (2019) proposed a deep learning approach for malware classification using unsupervised learning for malware detection. The proposed method extracted features from malware files using a convolutional auto-encoder neural network and then used clustering to group similar files. The experimental results demonstrated that the proposed method classified malware with high accuracy [50].

2.2.3 Feature Extraction

The process of identifying and selecting relevant characteristics, attributes, or features from raw data (such as binary code or system calls) that can be used to distinguish between malicious and benign software is referred to as feature extraction in malware detection. The selected features can then be fed into machine learning algorithms to train models that can correctly classify unknown samples as malicious or benign. Some of the previous works in this area are as follows.

R. K. Cunningham et al. (2010), based on byte-level file content statistical analysis, proposed a feature extraction technique for malware detection. Features such as byte frequency distribution, entropy, and n-gram frequency distribution are extracted using the proposed method. The features extracted are used in training a machine learning classifier for malware detection [19]. Y. Zhang et al. (2016) presented a feature extraction technique for malware detection with API call static analysis in their paper. The proposed method extracts features such as API call frequency, API call sequence, and API call argument values. The extracted features are used in training a machine learning classifier for malware detection [67]. S. Yadav et al. (2019) proposed a feature extraction technique for malware classification based on disassembly analysis of executable files. The proposed method extracts features like opcode frequency distribution, opcode sequence, and the presence of specific opcodes. The extracted features are used to train a malware classification machine learning classifier [60].

2.2.4 Deep Learning Methods

A branch of machine learning called deep learning uses multiple-layer artificial neural networks to discover and recognize patterns in huge and complicated datasets [27, 63]. Deep learning algorithms are used to automatically extract features from raw binary data or system call traces in the context of malware detection, eliminating the need for manual feature engineering. Before being utilized to identify fresh samples as harmful or benign, the neural networks are trained on a vast dataset of both benign and malicious samples. Deep learning has demonstrated promising results in malware detection, with some studies reporting detection rates of more than 99 percent [45].

There are many algorithms in deep learning for malware detection. Some of the most popular ones are as follows. For image categorization, Convolutional Neural Networks (CNNs) are commonly utilized, but they have also been used to detect malware, which function by detecting patterns in the binary code of malware samples [22]. Recurrent Neural Networks (RNNs) are built to work with sequence data, making them ideal for natural language processing and speech recognition. They've also been used to detect malware by analyzing the sequence of API calls made by the malware [30]. Generative Adversarial Networks (GANs) are a type of neural network that can generate realistic synthetic data, which have been used to generate synthetic malware samples for training [66].

Though machine learning approaches to malware detection are better than traditional methods, not every method provides expected results with perfection. When it comes to malware detection, the downsides of the above discussed methods are:

- Supervised learning: In the case of malware detection, supervised learning needs a large dataset with labeled data for training, which can be difficult to obtain. Furthermore, supervised learning models may be prone to over-fit on training data, limiting their ability to generalize to new, previously unseen malware [43].

- Unsupervised learning: It is based on the assumption that normal behavior can be distinguished from malicious behavior without prior knowledge of the malware. However, when it comes to malware detection, the line between normal and malicious behavior can be blurry, making it difficult to detect malware using unsupervised methods. In addition, unsupervised learning methods are prone to produce high number false positives [24].

- Feature extraction: Feature extraction is based on the assumption that certain aspects of malware behavior are more important for detection than others. However, identifying the most important features for malware detection can be difficult, and important features may be overlooked or missed [32].

- Deep learning models: They can be powerful malware detection tools, but they require a huge labeled dataset for training [15]. Deep learning models can also be computationally expensive to train and are prone to over-fit on training data [1]. Under the adversarial setting, these deep learning models can easily get evaded by adversarial attacks [17, 11, 13, 16, 12], which may degrade their performance on malware detection and disable their effectiveness.

The common element in all the above approaches is that they do not focus on relations of data samples. Leverage of these data relations might be trivial for some areas but not in malware detection. Because malware is a sophisticated and complex threat that has evolved over time. It is no longer a single file with a single purpose, but rather a network of files, functions, and behaviors that collaborate to achieve its objectives. To avoid detection by security software, malware can conceal its activities, encrypt itself, and use rootkits.

Researchers have turned to graph-based approaches for malware detection to combat these advanced techniques.

2.3 Graph-based Malware Detection

Graph-based approaches make use of the relationships between malware components such as files, processes, and network traffic. It is possible to analyze these relationships and detect patterns that indicate malicious activity by representing malware as a graph. Graph-based methods have been demonstrated to be effective in detecting both known and unknown malware, which have the potential to detect emerging threats.

A graph-based method to automatically analyzing malware behavior was introduced by S. Louafi et al. (2019). To represent the relationships between different events and actions during malware execution, the authors create causal graphical models. The models are trained on dynamic analysis data and used to forecast the behavior of previously unknown malware samples. The authors demonstrated the accuracy of their approach by testing it on a dataset of Android malware [42]. A graph-based approach to detect malware using machine learning is suggested by F. Lashgari et al. (2015). The authors portrayed the malware sample as a directed graph, with nodes representing code blocks and edges representing control flow. Using this representation, they take graph-based attributes and feed them into a machine learning algorithm. The authors demonstrated the accuracy of their method by evaluating it against a dataset of Windows malware samples [34]. H. Li et al. (2019) proposed a graph-based method for identifying malware variants. Each malware sample is represented by the authors as a graph, where nodes stand in for code fragments and edges for control flow. They learned a representation of the graph using graph convolutional networks, and then categorized the malware sample using this representation. The authors demonstrated that their method can effectively identify previously undiscovered variants by evaluating it against a dataset of Android malware variants [37].

However, the application of graph-based methods into malware detection generally attends to study the connections between various parts in a program, application or a file. It has been scarce to use these methods to work on more useful interactions among these programs, applications, and files. Some exceptions are that L. Chen et al. [10, 14] built graphs using file coexistence and used graph properties and belief propagation to perform malware detection; Y. Ye et al. [64] resorted to a heterogeneous graph upon different types of file relations to implement real-time Android malware detection. Despite these great successes, traditional graph-based approaches for malware detection have some disadvantages, which are detailed as follows:

- Restricted scalability: when working with large-scale graphs, some graph-based techniques may experience scalability problems. For large graphs, computing the eigen decomposition of the adjacency matrix is necessary for approaches based on matrix factorization, which can be computationally expensive. Graph neural networks, on the other hand, employ localized convolutional filters, which boosts their computational effectiveness and allows them to scale to bigger graphs [21].

- Complex graphs may be challenging for other graph-based methods to handle, particularly those with high-dimensional node characteristics or irregular architectures. GNNs, on the other hand, can work with complex graph architectures and can learn accurate node representations even from sparse and noisy data [52].

- Restricted feature extraction: some graph-based methods rely on manually created features or pre-established graph structures, which might not adequately capture the nuanced interactions between nodes. GNNs, on the other hand, have the ability to fully learn both the feature representations and the underlying graph structures.

- Lack of interpretability: it may be challenging to grasp how the model is making predictions using other graph-based approaches because of their lack of interpretability. By displaying the learnt feature representations and the graph topologies, GNNs can give interpretability, allowing researchers to learn more about the important features and relationships in the graph [35, 21].

Considering the facts that GNNs are more sophisticated in grasping the relations between the malware files and able to address the limitations yielded by traditional graph-based methods, in this thesis, we focus on designing GCN models (one of the most popular and significant GNNs) over the graph constructed on program data collection to perform malware detection.

Malware Detection using Enhancements of GCNs

In this chapter, we first introduce the issue of malware detection and outline how GCNs might be applied to it in Section 3.1. The process of building graphs to represent malware samples is covered in Section 3.2 along with the kinds of features that are frequently employed. Section 3.3 is concerned with the fundamental structure of graph convolutional networks and how it might be used to detect malware. We examine an extension of GCN that combines label propagation methods to enhance malware detection performance in Section 3.4. Section 3.5 introduces a further GCN version that makes use of residual connections to improve detection precision.

3.1 Synopsis

Malware, in order to accomplish its malevolent objectives just like any other software, is made up of a number of modules, functions, and instructions. For accurate malware analysis and detection, it is essential to comprehend the connections between these components because they are frequently intricately linked. Since they offer a potent method of representing the connections and dependencies that exist between various malicious components, graph structures are frequently linked to malware data.

Graphs offer a natural way to depict these linkages, enabling investigators to see the various malware components as nodes and the links between them as edges. They will learn more about how the virus functions, how it spreads, and what its main goals are as a result. Graph-based analysis, for instance, can be used to track the spread of malware throughout a network or to pinpoint (C&C) servers of a bot that a piece of malware is in communication with. Additionally, graphs can also be used to model a variety of other data types that are important for malware research, including file dependencies, network activity, system calls, and more. Analysts can better comprehend the virus and its behavior by merging these many sources of data into a graph-based model.

There are several graph-based techniques for malware detection such as Call Graph Analysis which focuses on analysing call-graph of a piece of malware to identify suspicious code patterns and vulnerabilities [18]. There is Control Graph Analysis which focuses on analysing control graph of a piece of malware code to find suspicious activity that can be linked to malware [65]. Similarly, there is data flow analysis which analyses data flow chart of piece of software to find out if it has data that has been altered by the infection or input from outside sources [38]. Behaviour analysis works by analyzing a piece of malware's behavioral graph, analysts can spot patterns of activity that point to malicious behavior, like efforts to increase privileges, change system settings, or connect to remote servers [56]. Network Graph Analysis by using a network graph, analysts can spot communication patterns that point to malicious activity, like attempts to get in touch with well-known command-and-control servers or other compromised sites [61].

In this thesis, we focus on using graph-based technique to analyze the relationship of the malware and its neighbors with close relationships to gain knowledge for malware detection. The structure of graphs that show the connections between various malicious programs, can be examined using GCN models. In other words, patterns can be spotted regarding the behavior that point to malicious activity by utilizing GCNs to learn characteristics from the graph data.

3.2 Graph Construction

In the case of malware detection using GCNs, creating a graph from the dataset regarding programs or applications is required, because GCNs are a sort of neural network designed to function on graph-structured data. A collection of nodes and edges can be used to represent graph-structured data, where nodes stand in for different types of entities, such as people, objects, or ideas, and edges signify the connections between those different entities. In the case of our malware detection, the nodes can represent several characteristics of a malware sample, like requests for permission or system calls, while the edges show the connections between these characteristics.

Considering that our data collection has no physical connections, a first and necessary step is to construct a graph to generate such connections. A strategy used frequently in graph-based machine learning applications is to create graph-type relations using a distance metric. As the features of our malware dataset, which would be introduced in detail in Section 4.1, are formalized as binary values that quantify the presence or absence of each feature, we use Hamming distance to search for the similarities and connections between the nodes of a malware dataset. This method involves creating an adjacency matrix that illustrates the similarity between malware samples. An adjacency matrix is a square matrix with entries that represent the connections between the graph's nodes in the rows and columns. The values in the adjacency matrix can indicate the similarity calculated using Hamming distance between sample features. For the constructed graph, the nodes can represent the various properties of a malware sample, and the edges represent the similarity of the characteristics of two samples that are measured by how near they are to one another.

After being created, the adjacency matrix and feature matrix can be passed into a GCN for analysis. The GCN propagates information along the graph by using a number of convolutional layers and the adjacency matrix as its input. This enables the GCN to consider the inter-dependencies between several aspects of a malware data nodes and train itself to spot patterns and correlations that are characteristic of malware, which results in more precise and effective malware detection.

However, given the size and abundance of attributes, the malware dataset produces a graph that is both dense and noisy, which accordingly leads to a significant amount of false negatives and false positives and may impair the model's accuracy. More specifically, when there are many connections between the nodes or edges in an adjacency matrix, it became challenging to comprehend the matrix and causes computational and performance problems for GCNs. That is, it is important to filter the adjacency matrix in the malware dataset to reduce its density and noise, and to increase the malware detection model's accuracy. To achieve the above said filtering, we implement two methods: one method used is the threshold method and another is the top k method.

3.2.1 Threshold Filtering

Threshold filtering of an adjacency matrix refers to the process of removing edges from the matrix that fall below a certain threshold value. In graph theory, an adjacency matrix is a square matrix that represents a graph by indicating which vertices are adjacent to each other. The entries in the matrix can be binary (indicating the presence or absence of an edge) or weighted (indicating the strength or weight of an edge).

By deleting the edges with low weights, threshold filtering can help to minimize the noise in the adjacency matrix. A specified criterion, such as a minimum weight or a minimum similarity score between two nodes, might be used to determine the threshold. To limit the amount of edges and make the network sparser, we could, for instance, define a threshold to exclude any edges with weights below a specific threshold value. Threshold filtering can make the graph easier to understand by minimizing the amount of edges and can aid in locating the relationships between the features that are most instructive. As a result, malware can be detected more accurately by GCNs.

3.2.2 Top-K Filtering

An adjacency matrix is subjected to top-K filtering, which entails choosing the K edges with the highest weights and eliminating all other edges. By concentrating exclusively on the most significant or robust connections between vertices, one might simplify a network in this manner. Top-K filtering, which keeps just the strongest K edges for each node, can aid in lowering the noise in the adjacency matrix. This method entails maintaining the top K edges with the highest weights and sorting the edge weights for each node in descending order. As a result, the adjacency matrix may become sparser, which may enhance the interpretability and precision of graph-based machine learning models for malware detection.

A specified criterion, such as the percentage of edges to keep or the threshold value of edge weights, can be used to determine the value of K. For instance, we can set K to be 200 of the total number of edges in the adjacency matrix if we only want to maintain the top 200 of edges with the highest weights. Top-K filtering can assist find the links between the attributes that are most informative by lowering edges between needed to be considered in the graph, which will make it easier to understand. As a result, malware can be detected in a more learning-effective yet cost-efficient way.

3.3 Graph Convolutional Networks

Malware creators are using increasingly stealthy methods to write the malware so that it can evade traditional methods. Malware frequently consists of a number of interconnected parts that work together to accomplish its evil objectives. These complex connections can be studied extensively if we can use graph-based structures to represent malware data.

With all the aforementioned disadvantages of traditional graph-based method, we can safely say that we need more sophisticated model that has the properties of learning from neighboring nodes and the ability to propagate that information. A particular class of neural networks made for graph data with the above purpose and qualities is called a Graph Convolutional Network (GCN). GCNs are able to function on data that has a more complicated and irregular structure, such as social networks, protein structures, or computer networks, in our case malware detection in contrast to typical Convolutions Neural Networks (CNNs), which are built for grid-like data like photographs.

GCN's fundamental premise is to extend the concept of convolutional filters from grid data to graph data. A GCN applies a learnable filter to each node in a graph, taking into consideration the features of the node and its neighbors, as opposed to applying a set filter to each pixel in a picture. GCNs can learn representations that accurately describe the nodes by iteratively propagating the node information through the graph [31]. By combining data from each node's neighboring nodes, the GCN changes each node's representation at each iteration. This makes it possible for the GCN to learn representations that accurately depict the graph's local structure. In malware detection scenario, assume, for instance, that application A in our graph is linked to applications B, C, and D. The GCN gathers feature information from nodes B, C, and D during the first iteration in order to update the representation of node A. By stacking multiple GCN layers, the model keeps updating the representation of node A in successive cycles.

Formally, we denote our constructed graph for malware detection as $G = (V, E)$, where V represents the programs (or applications) and E represents the connections (or similarities) between nodes, serves as the foundation for a GCN. In V, each node v_i has a corresponding feature vector x_i. The objective of the GCN is to train a function f that transfers these feature vectors to fresh feature vectors that record details about each node's immediate surroundings. The fundamental idea behind GCNs is to specify a filter that combines data from nearby nodes and applies it to the properties of the current node. For computing the new features for the current node, this filter can be seen as a matrix W that specifies how much weight should be assigned to each neighbor's features. The overview of a GCN model used for malware detection is illustrated in Figure 3.1.

The transformation of input data in a GCN is as follows:

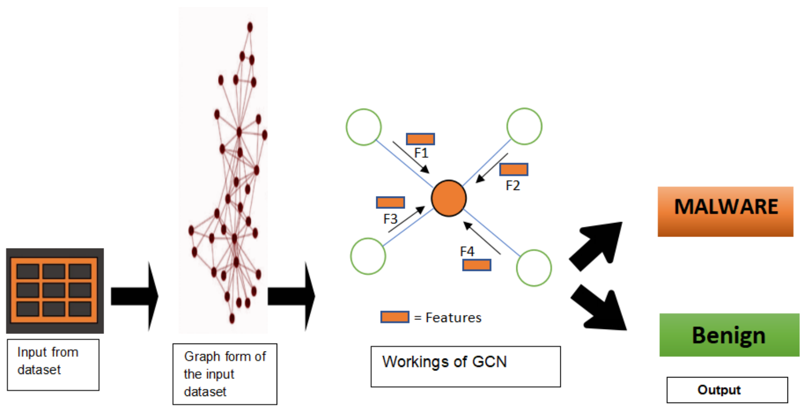

Figure 3.1: Overview of a Graph Convolution Network for malware detection

- Input graph: A graph $G = (V, E)$, where V represents the nodes and E represents the edges, is provided as its input with adjacency matrix A and feature matrix X.
- Initialization: For each node, the feature vector is set to $h_i^0 = x_i$.
- Aggregation: Using the filter matrix W, we aggregate data for each node v_i from its neighbors. This has the following mathematical representation:

$$h_i^l = \sigma\left(\sum_{j \in N(i)} W^l h_j^{l-1}\right) \tag{3.1}$$

where h_i^l represents the new feature vector for node i at layer l, σ represents an activation function similar to $ReLU$ or $sigmoid$, $N(i)$ represents the set of neighbors of node i, and h_j^{l-1} represents the feature vector for neighbor j at layer $l-1$.

- Transformation: After L layers of aggregation, we apply a transformation function g to each final feature vector h_i^L to obtain the final output y_i:

$$y_i = g(h_i^L) \tag{3.2}$$

where L is the number of layers in the GCN.

- Output: The final output y_i can be used for downstream tasks such as node classification or link prediction.

With several advantages for a GCN model, the main reasons why of using GCNs are best suited for malware detection are:

- Expressive power: As the graph is constructed on the malware dataset, GCNs can capture the intricate linkages and non-linear interactions between the malware features and learn expressive representations that effectively characterize the benign and malicious files by modeling the interactions between the many components of malware behavior as a graph structure. For example, GCNs can learn to recognize malware-specific patterns of behavior, such as a series of actions that results in sensitive information being accessed or shared without authority. With that knowledge, GCNs can reliably identify malware by learning these patterns of behavior, even if the infection has been updated to escape standard signature-based detection approaches.
- Robustness: Due to their operation on the graph structure, which can provide redundancy and contextual information, GCNs can be robust to noisy and incomplete data, and learn to generalize to new, previously unseen malware variants, which is especially useful for malware detection. This is crucial in malware identification since datasets such as the Drebin dataset can contain a large number of missing values or errors, and new variations emerge on a regular basis. GCNs have the ability to significantly increase the effectiveness of malware detection systems by providing a robust and scalable approach to malware identification.

Despite all the above said fine things about GCN, it still has few lacks. GCN has the restricted ability to generalize to novel or unseen nodes that weren't present in the training set is one possible drawback of GCNs. This is due to the fact that GCNs often execute convolutions by relying on the local structure of the graph, which may not adequately capture the community structure of the graph very effectively. The tendency of nodes in a graph to form groups or communities based on similar traits or characteristics is referred to as community structure. Because malware frequently exhibits similar behavior or features that can be classified together, community structure is significant in malware identification.

Another problem with GCNs in the area of malware detection is training them while they are deep. This is due to the fact that information can be diluted or lost as it passes through successive convolutional layers, making optimization harder and causing gradients to evaporate. This is known as the vanishing gradient problem, and it is a prevalent difficulty in deep learning. The vanishing gradient problem is more difficult in GCNs since they learn data representations by propagating information along the graph. If the gradients vanish as they propagate down the graph, updating the model's weights and improving its accuracy can be difficult.

To solve these potential problems of GCN, in this thesis, we propose enhancement to GCN using two different methods combined with GCN to improve its capability. To make GCN more generalizable to unseen data, we propose to use label propagation. To make GCN more susceptible for over-smoothing problem, we propose to use residual connections. We will discuss both the methods in detail in the following sections.

3.4 GCN+LPA

To perform malware detection, GCNs have a limited ability to capture global patterns and generalize to new or unknown nodes. This is due to the fact that GCNs rely on the graph's local structure, which may not always convey the overall link between nodes. Furthermore, GCNs' inability to generalize to new or unknown nodes may limit their performance in real-world circumstances where new malware variants develop on a regular basis. To address these shortcomings, we propose to integrate GCN with the Label Propagation Algorithm (LPA) to improve the models' accuracy.

LPA is a semi-supervised machine learning approach that is often utilized for classification tasks in which only a portion of the input is labeled. Based on the similarity of data points, the algorithm propagates labels from labeled to unlabeled data [41]. In the case of malware detection, for example, the labeled data points may represent known malware samples, while the unlabeled data points could be freshly discovered Android applications. LPA may infer the labels for previously encountered malware samples by assigning labels to unlabeled node points based on the labels of surrounding node points in the graph, making the GCN+LPA model more resistant against new types of malware. The overview of using GCN and label propagation for malware detection is illustrated in Figure 3.2.

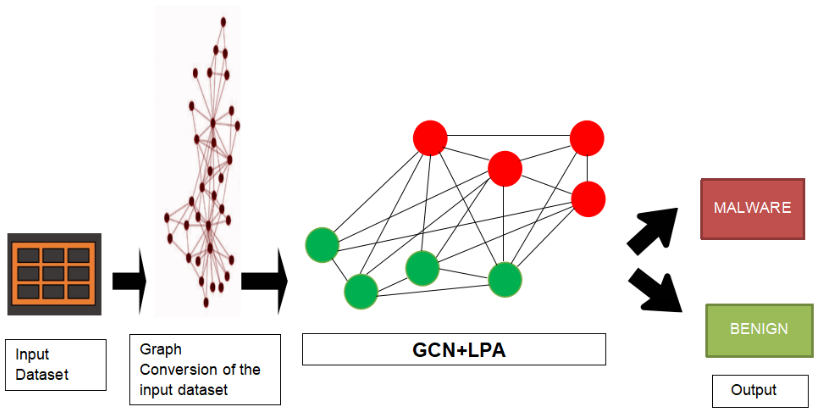

Figure 3.2: Overview of Graph Convolution Network+Label Propagation for malware detection

As a message-passing mechanism, the LPA spreads node label information through a graph's edges. With n being the number of nodes in the graph and $y(k)_i$ standing for the projected label distribution for node v_i in iteration k, let $Y(k) = [y(k)_1, ..., y(k)_n]$ be the soft label matrix in iteration k. In case there are no labeled nodes, or when $k = 0$, the initial label matrix $Y(0)$ consists of zero vectors. Alternatively, if there are unlabeled nodes ($I = 1, ..., m$), then one-hot label indicator vectors $y(0)_i$ are utilized [41]. With entries $d_{ii} = \sum_j a_{ij}$, let D be the diagonal degree matrix for A, where A is the adjacency matrix of the graph. LPA is then defined in iteration k as follows:

- $Y(k+1) = D^{-1}AY(k)$, where each row of $Y(k)$ is multiplied by A and normalized by D to propagate labels along edges.

- For all $y_i^{k+1} = y_i^0$ for all $i \leq m$, preserving labels for labeled nodes.

These two steps are repeated until convergence or a predetermined number of repetitions.

The answer for why use GCN+LPA and not just GCN, lies in the workings of GCN and GCN+LPA. While GCN excels at feature transmission, it lacks label information, which could be critical in distinguishing malicious from benign nodes. GCN does only the feature propagation and will leave out the information from labels. That is, the graph by GCN doesn't correlate the information from its neighboring nodes to whether they are malicious or benign. Leaving that information will not give the graph full picture of their neighboring nodes. This combination allows the GCN+LPA graphs to have a more complete image of their neighbors, resulting in a more effective approach to malware identification.

Specifically, we can start with a graph $G = (V, E)$, where V represents the collection of nodes and E represents the edges connecting nodes. Each node v_i in the graph is represented by a feature vector x_i. Additionally, we create a soft label matrix $Y(0)$ where $Y(0)_i = y(0)_i$ for $i \leq m$ and $Y(0)_i = 0$ otherwise and a set of labeled nodes $L = v_1, v_2, ..., v_m$ with known labels $y(0)_i$ for $I = 1, ..., m$. We set $A_{ij} = 1$ when there is an edge between nodes v_i and v_j; otherwise, we set $A_{ij} = 0$. This creates the adjacency matrix A. To calculate the degree matrix D, we utilize the summation of all elements in row I of A. We can now outline the stages that make up GCN+LPA:

The soft label matrix $Y(k)$ is updated using LPA in each iteration $k > 0$, as follows:

$$Y(k) = D^{-1}AY(k-1) \tag{3.3}$$

In this case, normalization by node degree is represented by $D - 1$, which is the inverse of D. The feature vectors $x_i(k)$ are modified using GCN in each iteration $k > 0$, as follows:

$$h_i(k) = \sigma(D^{-\frac{1}{2}}AD^{-\frac{1}{2}}x_i(k-1)W^{(k-1)}) \tag{3.4}$$

$$x_i(k) = h_i(k) \| x_i(k-1) \tag{3.5}$$

Here, $W^{(k-1)}$ is a weight matrix for layer k, and $||$ stands for concatenation. Up to convergence or the maximum number of iterations, steps 1 and 2 are repeated.

Using a classifier like logistic regression, the learned feature vectors $x_i(K)$ are then used to forecast labels for unlabeled nodes. How exactly is GCN+LPA used in malware detection is that the malware samples have been represented as graphs for the purpose of employing GCN+LPA to learn a representation of the graph that captures both structural and semantic data. Here is a description of how it functions:

- A graph is used to represent each malware sample, with nodes denoting API calls or other features and edges denoting connections between these features.
- The graph is created by deriving pertinent features from the malware sample's code after analysis.
- Depending on the matching API call or other feature, the feature vectors for each node are initialized.
- Based on known malicious or benign samples, labeled nodes are found, and their labels are utilized to initialize the soft label matrix $Y(0)$.
- To learn a representation that includes both structural and semantic details about the malware sample, GCN+LPA is applied to the graph.
- The newly discovered, unlabeled samples are categorized as malicious or benign using the learnt representation.

The results for this proposed method are showcased in Section 4.5.

3.5 GCN+Residual

Over-smoothing occurs in GCNs when too many layers of convolutional operations are applied to the input graph, resulting in loss of discriminative power and blurring of learned node features. Essentially, the representations of nodes in the graph become too similar across classes, making differentiation difficult for the model, which lead to poor performance in tasks like node classification and link prediction, which include our malware detection application.

How does over-smoothing impact on malware detection can be understood in the following example. Assume we have a dataset of malware samples, with each sample represented as a fixed-length binary sequence. Each binary sequence can be depicted as a graph, where each node corresponds to a byte and is linked to its adjacent nodes. We can then use a GCN to classify malware samples by learning features from the graph representation. If we apply too many GCN layers, the model may over-smooth the bytes' features, making it difficult to distinguish between benign and malicious samples. Some malware, for example, may use the same byte patterns as legitimate software, and if the model over-smooth the features, the malware may be incorrectly classified as benign.

To overcome this major issue using GCN for malware detection, we are taking help from residual connections, which include two effective techniques: initial residual and identity mapping. We detail them as follows.

- Initial Residual: The initial residual connection is a method of propagating the graph's original input features through the GCN layers. This is accomplished by first adding the initial features to the GCN layer output before applying the activation function. At each layer l, the initial residual technique constructs a skip connection from the input feature matrix $X^{(0)}$ to the output feature matrix $H^{(l+1)}$ as follows:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)} + X^{(0)}W_0 + H^{(0)}) \tag{3.6}$$

where $X^{(0)}W_0 + H^{(0)}$ is a learnable parameter that adds the initial residual to the output of each layer. This technique helps to preserve the original node features and improve information flow across layers. The initial residual connection aids in the preservation of the original information in the graph and ensures that the model can learn useful representations even if the graph structure is sparse.

- Identity mapping: The identity mapping is a simpler method for connecting the GCN layers without introducing any additional parameters. It entails directly adding the input features to the GCN layer's output, bypassing any nonlinear transformations. At each layer l, the identity mapping technique adds an identity matrix to the weight matrix $W^{(l)}$ as follows:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)} + X^{(0)}W_0 + H^{(0)}) + H^{(l)} \tag{3.7}$$

  where $H^{(l)}$ is added to the output of each layer via element-wise addition. This technique helps to prevent over-smoothing by preserving high-frequency information in node representations. When the input features are already sufficiently informative, and the model only needs to learn a few additional features to make accurate predictions, identity mapping is useful. It is also beneficial in avoiding over-smoothing and improving the training process stability.

By incorporating the techniques mentioned above, we present a solution to address the over-smoothing issue with GCNs by combining the aforementioned techniques. This method is referred to as GCN+residual in this thesis. The overview of using GCN and residual for malware detection is illustrated in Figure 3.3.

The proposed method uses the initial residual connection at each layer, resulting in a skip connection from the input layer to the layer's output. Furthermore, the identity matrix is combined with the weight matrix, which improves the learning process and allows for

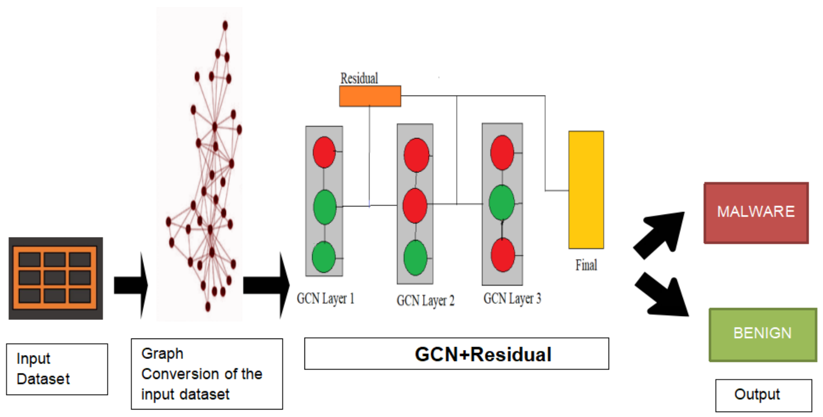

Figure 3.3: Overview of a Graph Convolution Network+Residual for malware detection

better feature representation improving the task of node classification in malware detection. We hope to mitigate the problem of GCN over-smoothing, which is known to degrade performance in the task of node classification, by using this approach [9].

The GCN+residual model defines the following propagation rule:

$$H^{(l+1)} = \sigma((1-\alpha^{(l)})\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)} + \alpha^{(l)}H^{(0)} + H^{(l)}) \tag{3.8}$$

where $H(l)$ is the output feature matrix at layer l, $H^{(l+1)}$ is the output feature matrix at layer $l+1$, $W^{(l)}$ is the weight matrix of layer l. $\tilde{A} = A + I$ is the adjacency matrix of the graph, where I is the identity matrix. $\tilde{D}$ represents degree matrix of the graph, calculated as $\tilde{D}ii = \sum j\tilde{A}_{ij}$. $ReLU$ is the rectified linear unit activation function. The residual connection $H^{(l)}$ allows information from previous layers to be retained and incorporated into the next layer's output.

This model effectively captures the graph's local and global structural information and learns more informative features for downstream tasks like node classification and link prediction, which also include our malware detection applications. The results are showcased in Section 4.5.

Experimental Results and Analysis

In this chapter, we test our two models Graph Convolution Networks with Label Propagation and Graph Convolution Networks with residual against a real malware data set. First, we describe the data set in detail, followed by the experimental parameters. Following that, we assess the effects of various parameters to find the effectiveness of GCN enhancements on malware data set.

4.1 Dataset

For our experiment, we utilized an Android malware dataset called ”Drebin”. Android malware also known as mobile malware refers to malicious software that targets mobile phones and Personal Digital Assistants (PDAs) with wireless capabilities, causing system failure and the loss or disclosure of sensitive information. Keeping wireless phone and PDA networks safe and secure from electronic threats like viruses or other malware has grown more difficult as a result of their increased use and sophistication.

Dataset used is taken from Kaggle an open source platform for datasets and kernels for data science [1]. The Drebin dataset, which was employed in this work, has 215 feature vectors taken from a sample of 15,036 mobile applications. This sample contained 5,560 instances of malicious programs obtained from the Drebin project, as well as 9,476 examples of benign applications. Out of 215 unique feature vectors, 74 of them are API call

[1] https://www.kaggle.com/datasets/shashwatwork/android-malware-dataset-for-machine-learning

Table 4.1: Android Malware Dataset Statictics

Dataset	**No. of Entries**	**No. of Benigns**	**No. of Malwares**	**No. of Features**
Drebin	15,036	9,476	5,560	215

signatures, 113 of them are manifest permissions, 23 are intent vectors, 6 are command signatures and 1 is class label that labels whether the respective file is benign or malware.

The features as said above are API call signatures, Manifest permissions, intent vectors and command signatures.

- API call signatures: They refers to the Android application's unique set of API calls. These signatures aid in the identification of specific patterns of behavior associated with malicious software. Some of the important API call signatures.
  - Android's **onServiceConnected** and **bindService** methods are used to connect to a remote service. Malware may employ these techniques to communicate with a remote server or to carry out other malicious actions.
  - **attachInterface** is an Android method for attaching a communication interface to a Binder object. Malware may use this method to communicate with other Android system components.
  - In Android, the **ServiceConnection** interface is used to monitor the connection to a service. This interface may be used by malware to connect to and maintain a connection with a remote server.
  - **android.os.Binder** is an Android class that facilitates interprocess communication. This class may be used by malware to communicate with other processes or components of the Android system.
  - **getBinder** is an Android method for obtaining a Binder object. Malware may use this method to obtain a reference to a Binder object and use it to communicate with other Android system components.

- **ClassLoader** is a Java class that is used to dynamically load classes during runtime. This class may be used by malware to load additional code or to execute malicious code at runtime.
- **HttpGet.init** is a method in Android that is used to start an HTTP GET request. This method may be used by malware to establish network connections and communicate with remote servers.
- **SecretKey** is an Android class that represents a secret key. Malware may employ this class to encrypt or decrypt data, or to carry out other malicious actions that necessitate the use of secret keys.

* Manifest permissions: To access certain features and data on the device, Android applications require specific permissions. An Android application's manifest file contains a list of all the permissions that the application requires.

  - Applications can install packages automatically with **INSTALL_PACKAGES**. This privilege could be used by malware to install other malware or undesirable applications on the device.
  - **CAMERA** grants access to the device's camera to an application. Without the user's knowledge or permission, malware might utilize this permission to take photos or videos.
  - An application's ability to change an account's sync settings is provided by **WRITE_SYNC_SETTINGS**. This ability could be used by malware to change the device's synchronization settings, which could lead to data loss or other problems.
  - An application's ability to read a user's bookmarks and browsing history is **READ_HISTORY_BOOKMARKS**. This permission could be used by malware to steal sensitive data, including login credentials or other private information.

  - An application can access the Internet thanks to **INTERNET**. This capability could be used by malware to communicate with a remote server and carry out nefarious tasks like data theft or spam distribution.
  - An application can communicate with NFC tags or other gadgets using **NFC**. This permission could be used by malware to obtain private data from NFC tags or other gadgets.
  - With **ACCESS_LOCATION_EXTRA_COMMANDS** flag an application can access advanced location settings. This capability could be used by malware to locate the device and follow the user's movements.

- Intent vectors: The Android operating system's core feature, intentions, enables communication between programs and the system. An android application's use of intents and their accompanying targets are represented by intent vectors. In order to communicate with other programs or services on the device and carry out harmful activities, malware may exploit various intent vectors.
  - When the device has finished starting up, the android operating system initiates the intent action **android.intent.action.BOOT_COMPLETED**. When an app receives this intent, it can take action by starting background services or configuring alerts, among other things.
  - When a device is plugged into a power source, an intent action called **android.intent.action.ACTION_POWER_CONNECTED** is launched. When the device is charging, malware may leverage this purposeful action to carry out nefarious tasks like data theft or malware installation.
  - **android.intent.action.PACKAGE_REMOVED** is triggered,when an app is removed from the device. When an app is deleted, malware may utilize this intent action to carry out nefarious tasks like destroying crucial data or changing system settings.

- **android.intent.action.ACTION_POWER_DISCONNECTED** is launched when a device is unplugged from a power source, an intent action called . When the device is not charging, malware may use this intent action to carry out nefarious tasks like transmitting spam or stealing data.
- The intent action **android.intent.action.PACKAGE_ADDED** is triggered when an app is installed on a device. When an app is installed, malware may utilize this intended action to carry out nefarious deeds like installing further malware or changing system settings.
- The intent action **android.intent.action.ACTION_SHUTDOWN** is triggered as the device is shutting down. When the device is going off, malware may utilize this deliberate action to carry out nefarious tasks like erasing crucial data or changing system settings.
- When an app's data is deleted from the device, an intent action called **android.intent.activity.PACKAGE_DATA_CLEARED** is launched. When an app's data is cleansed, malware may utilize this purposeful action to carry out nefarious tasks like installing more malware or changing system settings.
- When the package information for an app changes, an intent action called **android.intent.action.PACKAGE_CHANGED** is launched. When an app's package information is altered, malware may utilize this intent action to carry out harmful operations like changing system settings or adding new malware.
- To send a message to a particular recipient, **android.intent.action.SENDTO** is used. This purpose action may be used by malware to send spam or other unwanted messages to the contacts on the device.
- **android.intent.action.SCREEN_ON** is an intent action that begins as soon as the screen of the device is activated. When the device is in use, malware may utilize this intent action to carry out nefarious tasks like data theft or system

setting modification.

- Command Signatures: The distinct collection of commands or shell commands that the Android application uses are referred to as its ”command signature” in this functionality. Malicious software may employ particular commands to carry out specific tasks, such accessing private files or running system commands.
  - Linux-based operating systems, such as Android, use the **mount** command to mount a file system. This command may be used by malware to access private files or to change the file system in some other way.
  - In Linux-based operating systems, the **chmod** command is used to modify a file or directory’s permissions. This command may be used by malware to change a file’s or directory’s permissions so that it can access higher-level resources or avoid detection.
  - In Linux-based operating systems, the **remount** command is used to remount a file system as read-write or read-only. This command may be used by malware to change the file system in some way or to hide by making files read-only.
  - In Linux-based operating systems, the **chown** command is used to change the owner of a file or directory. This command may be used by malware to change the ownership of files in order to avoid detection or to take control of sensitive files.
  - The Android file system’s **/system/bin** directory houses all of the necessary system executables and binaries. This directory may be targeted by malware in order to acquire root access or make changes to the system.
  - The Android file system has a directory called **/system/app** that houses the device’s pre-installed system apps. This directory may be targeted by malware to alter or replace system apps or to access private data kept by these apps.

4.2 Parameters

In this experiment, several parameters were used in executing the GCN models in the experiment. The selection of appropriate parameters is critical for the model to effectively learn from the data and generalize to previously unseen data. These parameter settings are specified as follows:

- The initial learning rate, which controls the step size used during optimization to minimize the loss function, was the first parameter. In this experiment, the learning rate is set to 0.01 to imply that the optimizer takes larger steps in the beginning to quickly converge towards the optimal solution.
- During training, the batch size parameter determines the number of instances that are processed in each forward and backward pass. In this experiment, a batch size of 42 was chosen, which is a commonly used value in training neural networks.
- Another parameter that regularizes the model to avoid overfitting is weight decay. The weight decay parameter is set to 1e-6, indicating the strength of the weight regularization penalty.
- Finally, the hidden layers determine the model's complexity and ability to learn complex patterns in data. In this experiment, the GCN model has 16 hidden layers, which is a relatively deep architecture.

4.3 Baselines

To evaluate the effectiveness of Graph Convolutional Networks (GCNs) and its enhancements for malware detection, we employed four baseline machine learning techniques. These methods are widely used in malware detection and serve as a basis for comparison.

We can determine the effectiveness of GCNs for this task by comparing their performance to that of these algorithms. The machine learning algorithms we used as our baseline are:

- **Support Vector Machines (SVMs)**: Due to its capacity to handle high-dimensional feature spaces and their tolerance to noisy data, SVMs are a popular choice for malware detection [29].
- **Neural Networks**: Because of its capability to learn complicated patterns and correlations in data, neural networks have demonstrated promising results in malware detection [7].
- **Random Forest**: Random Forest is a common ensemble learning technique that can be used for malware detection categorization jobs. It constructs numerous decision trees from randomly selected subsets of features and data samples, then combines their outputs to produce predictions [4].
- **Naive Bayes**: Naive Bayes is an algorithm functioning on the probability that can be used for malware classification jobs. Based on a sample's features, the likelihood that it belongs to a particular class is calculated [51].

4.4 Evaluation of Graph Construction Methods

We experimented with two different approaches to filter the adjacency matrix in order to obtain a better graph for malware identification.

- For the threshold filtering method, after careful observation of the initial adjacency method with out filtering, initially we experimented with $>.7$ and $>.8$ but resulted to below average results that are somewhere near 60%. Then we moved to the values that are $>.90$, $>.95$, $>.97$. These three gave the best results. Results are shown in Table 4.2.

- For Top-K method, keeping in mind the size of the adjacency matrix which is 15030 X 15030 we wanted to filter the most important edges. In that process after careful consideration we chose to filter it in top-20, top-50 and top-100. These results showed this type of filtering is not helpful as the results are between 60% to 75%. Results are shown in Table 4.2

Table 4.2: Graph Construction Methods Evaluation

Method	**Values**	**Accuracy**
Threshold	>.90	91.59
	>.95	92.14
	>.97	87.89
Top-K	Top-20	73.06
	Top-50	64.59
	Top-100	67.39

4.5 Evaluations of GCN and Its Variants

By choosing the filtering method as threshold filtering and the values chosen as threshold is values >.95, we evaluate in this section which compares the performance of three graph convolutional network (GCN) models for malware detection.

- The first model is a traditional GCN that just propagates node features through the graph. This predicted the accurate results of benign or malware with an accuracy of 92.14. The GCN is fed a matrix representation of the code graph, with each row representing a node in the graph and the columns representing the node's features. The GCN then conducts a series of graph convolutional operations to this input, updating the feature vectors of the nodes depending on the features of their nearby nodes with each operation. Then outputs an accuracy of the classification done by GCN as benign or malware.

- The second model is a GCN with a Label Propagation Algorithm or LPA. The LPA or Label Propagation Algorithm is utilized in the case of GCN+LPA to transmit labels from nodes that are labeled to unlabeled nodes. LPA operates by labeling nodes based on the neighboring labels. This means that nodes that are similar to labeled nodes are apparent to have the same label assigned to them. The GCN is then utilized to spread feature information over the graph. This means that surrounding node features are integrated to form a feature representation for each node. We were able to achieve an accuracy of 94.02.

- GCN+residual is the third model, which uses residual connections to alleviate the over-smoothing issue. The GCN+residual design has a skip connection that connects a layer's input to its output. This enables the network to learn residual connections, which are variations between a layer's input and output, making it easier to understand more complex data patterns. This predicts whether the file is malware or benign with great accuracy.

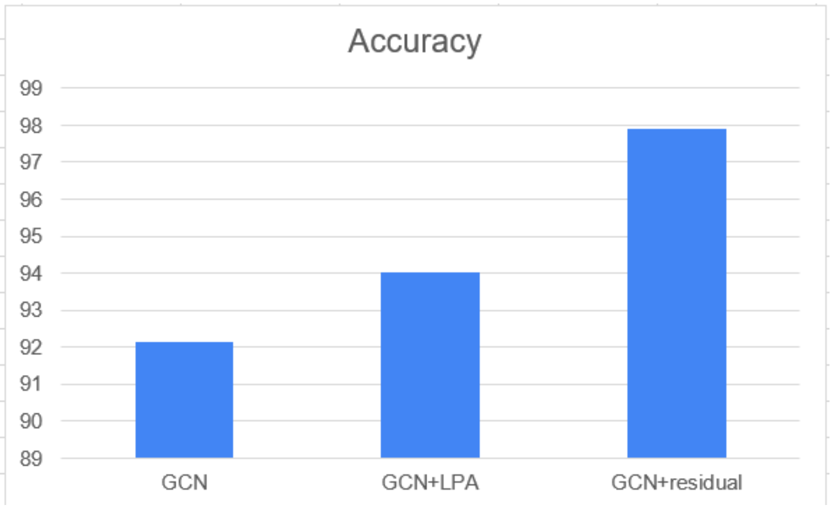

Figure 4.1: Comparison between GCN models

The above discussed results are displayed in Figure 4.1. Section 4.7 will give an in depth analysis of the methods and their performance.

4.6 Comparison with Baselines

In this section, we did a comparative analysis with numerous baseline methods that are regularly used in the field to evaluate the performance of all the three models, GCN, GCN+LPA, GCN+residual for malware detection. To that goal, we developed a benchmark dataset to assess each model's accuracy in detecting malware. The comparative results are reported in Table 4.3. Our GCN-based models outperform established baseline methods, according to the results, which are presented in the form of a performance comparison table. The GCN model, in particular, scored 92.14% accuracy, while the GCN+LPA model and GCN+residual model achieved even higher accuracy ratings of 94.02% and 97.89%, respectively. This implies that our proposed models can greatly enhance malware detection accuracy, particularly when compared to established traditional approaches such as SVM, Neural Network, Random Forest, and Naive Bayes without using any structural information. The results will be analysed in detailed in Section 4.7.

Table 4.3: Comparision between Baselines and GCN and its enhanced versions

Models	**Accuracy**
SVM	88.10
Neural Network	88.79
Random Forest	90.39
Naive Bayes	89.62
GCN	92.14
GCN+LPA	94.02
GCN+Residual	97.89

4.7 Analysis

It is critical to compare and analyze the findings of various Graph Convolutional Network models for malware identification after analyzing their performance in the previous sections. In this section, we will analyse in detail about the results gained from the previous section's experiments. In terms of accuracy and we will compare the baselines to the GCN, GCN+LPA, and GCN+residual models.

- First we will look into the graph construction methods. Both the methods are performed on the Drebin dataset and the results are shown in Table 4.2. Top-K performed poorly when compared to threshold method. Not only the Top-k values showcased but we also tried more values such as Top-10 and Top-200. But they resulted in crashing of the program leaving with no proper output. Where as the threshold performed very well with all the values. As mentioned in the previous sections some of the values that are >.90 did not perform well. But we achieved best results with the results showcased in Table 4.2. The threshold of .95 is selected for reason of its out performance out of all the other values. The same threshold is used through out the experiment.

- Secondly we talk about comprehensive comparison between the baselines and GCN enhancements in this section. The results in Table 4.3 demonstrate that GCN and its enhancements outperform all of the traditional machine learning algorithms currently being used in detecting malware. Even though the four baseline algorithms are widely used in this field, our GCN achieved a significant improvement in accuracy. This indicates that the emphasis on relations between nodes in a graph-based malware detection approach leads to more sophisticated results than traditional machine learning methods. It should be noted that some baseline algorithms may perform well with specific graph-based datasets, but it is not feasible to obtain rapidly updated malware datasets in real-time in a graph structure. Therefore, there is a need for methods that can construct their own graph and still outperform traditional baseline methods. GCN effectively addresses this challenge, making it a promising path for malware detection.

- The performance findings of all the three GCN enhancements, GCN, GCN+LPA, and GCN+Residual, show that focusing on relationships in malware detection via GCNs increases malware classification accuracy. The GCN model was 92.14% accurate, while the GCN+LPA model was 94.02% accurate. With an accuracy of 97.89%, the GCN+Residual model outperformed both. These findings suggest that adopting graph-based models, such as the GCN+LPA and GCN+Residual models, may yield more nuanced results than typical machine learning techniques. The improved performance achieved by adding residual connections and the LPA method demonstrates the efficiency of using the power of the graph structure to improve malware classification. These findings show that GCN upgrades are promising options for malware identification.

Challenges

Throughout the course of this experiment, a number of obstacles were encountered that made it extremely difficult to produce the required results. The size of the Drebin dataset was one of the main issues that were encountered. Due to the overwhelming amount of data, processing took a long time and specialist technology was required to handle the load. Additionally, the experiment's adjacency matrix was exceptionally big, measuring 15030 by 15030. This led to a lot of crashes and errors during computation, making it challenging to get precise results quickly.

Furthermore, creating a graph from such a huge dataset was challenging in and of itself. Prior to creating the graph, the data needed to be cleansed and pre-processed. This pre-processing step needed proficiency in managing and modifying huge datasets as well as familiarity with the numerous methods for cleaning and putting together data. Furthermore, creating a graph from such a huge dataset needed a significant amount of computational power and specialized software. Numerous intricate calculations and algorithms were used to form the graph, which increased processing time and resource needs. Overall, handling the dataset's size and complexity was a difficult undertaking that called for specific knowledge and equipment successfully perform the experiment.

Additionally, the resource-intensive nature of the graph construction and GCN software presented another issue in addition to the difficulty of handling huge datasets. High computational power was needed to run the GCN model and subsequent improvements, which is not always readily available. Furthermore, the experimentation and optimization processes take a lot of time due to the lengthy execution times for these algorithms. For researchers or practitioners who lack access to powerful hardware or who cannot afford the computational resources necessary to run these models, this can be a hurdle.

Future Work

One subject for future work is to further enhance the performance of GCN models for malware detection. There is certainly opportunity for improvement even if these models provided encouraging outcomes in the current experiment. To identify the best values for the particular datasets, one strategy is to experiment with various hyper parameters. Another strategy to increase the precision of the adjacency matrix is to further optimize the graph creation procedure. To enhance the overall performance of the models, we might also look into other methods like attention processes or several layers of neural networks.

Another direction is to look into how the effectiveness of the GCN models depends on the adjacency matrix's accuracy. The adjacency matrix was created in the current experiment using approaches that are currently used to form graphs. However, it might be possible to increase the method's accuracy for creating graphs. For instance, to enhance the adjacency matrix's quality, we could investigate the usage of various network creation algorithms or pre-processing methods. In order to better convey the complexity of malware interactions, we may also look into the use of different graph representations, such as attributed graphs or heterogeneous graphs. We will be able to enhance the performance of the GCN models and enable more precise malware detection by enhancing the quality of the graph creation process.

Testing the models on various malware datasets is crucial to ensuring their generalizability. This would allow us to find any flaws or restrictions in the models and help us assess whether the models function consistently across various virus kinds. By testing the models on various datasets, we can also assess whether any adjustments or enhancements are required to ensure their efficacy in a variety of situations.

Conclusion

In this thesis, instead of detecting malware using traditional machine learning models on the individual programs or applications, we take a further step to construct graph to connect benign and malicious samples and leverage GCNs to perform malware detection. To address the limitations of vanilla GCNs, we propose two enhancement versions, including devising label propagation and residual connections into GCNs to improve the detection performance. The experimental results demonstrated that, with an accuracy of 97.89%, the GCN+Residual approach outperformed both the conventional machine learning baseline algorithms and the other GCN versions. It is crucial to remember that creating a graph and applying GCN and its improvements to the dataset generate a lot of benefits but require a lot of effort and resources as well. Due to the magnitude of the dataset and the difficulty of creating the adjacency matrix, the experiment encountered difficulties. In addition to malware detection, these models can be used for other node classification tasks over graphs, demonstrating their versatility and promise.

Bibliography

[1] Haider Abbas, Suleman Khan, and Khaled Salah. Deep learning for malware detection: A survey. *arXiv preprint arXiv:1801.00318*, 2018.

[2] Muhamed Fauzi Bin Abbas and Thambipillai Srikanthan. Low-complexity signature-based malware detection for iot devices. In *Applications and Techniques in Information Security: 8th International Conference, ATIS 2017, Auckland, New Zealand, July 6–7, 2017, Proceedings*, pages 181–189. Springer, 2017.

[3] Muhammad Shoaib Akhtar and Tao Feng. Malware analysis and detection using machine learning algorithms. *Symmetry*, 14(11), 2022.

[4] Mohammed S Alam and Son T Vuong. Random forest classification for detecting android malware. In *2013 IEEE international conference on green computing and communications and IEEE Internet of Things and IEEE cyber, physical and social computing*, pages 663–669. IEEE, 2013.

[5] Mohammad A AlZu'bi and Mohammad A Alsmirat. Malware detection using machine learning techniques: A survey. *Journal of Information Processing Systems*, 15(5):1075–1094, 2019.

[6] Ömer Aslan Aslan and Refik Samet. A comprehensive review on malware detection approaches. *IEEE Access*, 8:6249–6271, 2020.

[7] Chris M Bishop. Neural networks and their applications. *Review of scientific instruments*, 65(6):1803–1832, 1994.

[8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Behavior-based malware detection using data mining. *Data Mining and Knowledge Discovery*, 18(1):63–94, 2009.

[9] Jianfei Chen, Jun Zhu, and Le Song. Simple and deep graph convolutional networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 14492–14499, 2020.

[10] Lingwei Chen, William Hardy, Yanfang Ye, and Tao Li. Analyzing file-to-file relation network in malware detection. In *Web Information Systems Engineering–WISE 2015: 16th International Conference, Miami, FL, USA, November 1-3, 2015, Proceedings, Part I 16*, pages 415–430. Springer, 2015.

[11] Lingwei Chen, Shifu Hou, and Yanfang Ye. Securedroid: Enhancing security of machine learning-based detection against adversarial android malware attacks. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 362–372, 2017.

[12] Lingwei Chen, Shifu Hou, Yanfang Ye, and Lifei Chen. An adversarial machine learning model against android malware evasion attacks. In *Web and Big Data: APWeb-WAIM 2017 International Workshops: MWDA, HotSpatial, GDMA, DDC, SDMA, MASS, Beijing, China, July 7-9, 2017, Revised Selected Papers 1*, pages 43–55. Springer, 2017.

[13] Lingwei Chen, Shifu Hou, Yanfang Ye, and Shouhuai Xu. Droideye: Fortifying security of learning-based classifier against adversarial android malware attacks. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 782–789. IEEE, 2018.

[14] Lingwei Chen, Tao Li, Melih Abdulhayoglu, and Yanfang Ye. Intelligent malware detection based on file relation graphs. In *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015)*, pages 85–92. IEEE, 2015.

[15] Lingwei Chen, Xiaoting Li, and Dinghao Wu. Adversarially reprogramming pretrained neural networks for data-limited and cost-efficient malware detection. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, pages 693–701. SIAM, 2022.

[16] Lingwei Chen and Yanfang Ye. Secmd: make machine learning more secure against adversarial malware attacks. In *AI 2017: Advances in Artificial Intelligence: 30th Australasian Joint Conference, Melbourne, VIC, Australia, August 19–20, 2017, Proceedings 30*, pages 76–89. Springer, 2017.

[17] Lingwei Chen, Yanfang Ye, and Thirimachos Bourlai. Adversarial machine learning in malware detection: Arms race between evasion attack and defense. In *2017 European intelligence and security informatics conference (EISIC)*, pages 99–106. IEEE, 2017.

[18] Mihai Christodorescu, Somesh Jha, Sanjit Seshia, Dawn Song, and Randal Bryant. Semantics-aware malware detection. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 32–46. IEEE, 2005.

[19] Robert K Cunningham and Philip KP Chan. Malware detection using statistical analysis of byte-level file content. In *2010 Sixth International Conference on Information Assurance and Security*, pages 178–183. IEEE, 2010.

[20] Dorothy E Denning and Ira J Heavens. Intrusion detection using variable-length anomaly signatures. *Proceedings of the 7th ACM conference on Computer and Communications Security*, pages 67–76, 2001.

[21] S. M. Fayyaz, Z. Wang, M. Abdullah-Al-Wadud, and M. Alazab. A survey of graph-based malware detection techniques. *IEEE Access*, 8:23528–23544, 2020.

[22] Sebastian Garcia and Stefan Jacoby. Malware traffic detection using convolutional neural networks. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pages 696–701. IEEE, 2018.

[23] Markus Goldstein, Thomas Minka, and Rosalind Picard. Anomaly detection using behavior-based models. *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX)*, pages 309–316, 2000.

[24] Ali Z Gondal, Mamoun Alazab, Michael Hobbs, and Jemal Abawajy. Malware detection and machine learning: limitations and prospects. *Journal of Computer Virology and Hacking Techniques*, 15(1):1–15, 2019.

[25] Kyungtae Han, Donghyun Kim, and Junyoung Kim. A heuristic approach to malware detection in embedded systems. *IEEE Access*, 5:15552–15562, 2017.

[26] Sarmadullah Haq, Areeba Shahid, Ali Raza, Muhammad Waheed, Tahir Mahmood, and Muhammad Imran. Feature selection for machine learning-based malware detection: A comparative study. *Computers & Security*, 93:101804, 2020.

[27] William Hardy, Lingwei Chen, Shifu Hou, Yanfang Ye, and Xin Li. Dl4md: A deep learning framework for intelligent malware detection. In *Proceedings of the International Conference on Data Science (ICDATA)*, page 61. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2016.

[28] Rafiul Islam, Md Saiful Islam, Md Sabbir Ahmed, and Mohamed Nasser. Unsupervised learning techniques for malware traffic analysis. In *2019 21st International*

Conference on Computer and Information Technology (ICCIT), pages 1–6. IEEE, 2019.

[29] Vikramaditya Jakkula. Tutorial on support vector machine (svm). *School of EECS, Washington State University*, 37(2.5):3, 2006.

[30] Taeho Kim and Sungzoon Yoon. Malware detection using long short-term memory networks. *arXiv preprint arXiv:1605.09039*, 2016.

[31] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[32] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Limitations of static feature extraction in malware detection: A cautionary note. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 172–191. Springer, 2016.

[33] Jyoti Landage and MP Wankhade. Malware and malware detection techniques: A survey. *International Journal of Engineering Research*, 2(12):61–68, 2013.

[34] Ahmad Daneshmand Lashgari and Ali A Ghorbani. Malware detection using graph-based analysis and machine learning. In *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, pages 27–34. IEEE, 2015.

[35] N. T. Le, T. T. Dinh, D. H. Hong, and T. T. Nguyen. Malware classification with convolutional neural networks using graphical representations. *Journal of Computer Science and Cybernetics*, 34(1):13–28, 2018.

[36] Sangwon Lee, Junho Lim, and Byung-Gyu Kang. A heuristic-based malware detection framework for android using api call sequences. *Sensors*, 17(11):2663, 2017.

[37] Hui Li, Jing Li, and Yu Li. Detecting malware variants with graph convolutional networks. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 2683–2690. IEEE, 2019.

[38] Wenjuan Li, Aziz Mohaisen, and Yang Xiang. Malware graph based detection using static and dynamic analysis. *Computers & Security*, 63:146–162, 2016.

[39] Richard P Lippmann, David J Fried, Isaac Graf, James W Haines, Karla Kendall, David McClung, and Del Weber. A comprehensive study of the detection of anomalous system calls using machine learning. *IEEE Transactions on software engineering*, 29(6):524–535, 2003.

[40] Hu Liu, Hui Cheng, and Shouzhi Zhang. Unsupervised malware detection based on behavioral characteristics. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 931–936. IEEE, 2017.

[41] Yiding Liu, Yufei Chen, Jun Liu, and Xiang Zhang. Gcn-lpa: Multi-layer feature propagation for malware detection. In *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 1–6. IEEE, 2019.

[42] Sihem Louafi, Frédéric Guérin, and Arnaud Salaün. Towards automated dynamic malware analysis using causal graphical models. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 251–258. IEEE, 2019.

[43] Wei Lu, Junbo Liang, Chunhua Liu, Yijia Cao, Yibo Zhang, and Jun Chen. Malware classification with graph convolutional networks. *arXiv preprint arXiv:1904.00373*, 2019.

[44] Avijit Mondal, Subrata Paul, Anirban Mitra, and Biswajit Gope. Automated signature generation for polymorphic worms using substrings extraction and principal compo-

nent analysis. In *2015 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, pages 1–4. IEEE, 2015.

[45] Lakshmi Nataraj, Sundaramoorthy Karthikeyan, and Greg Jacob. Deep learning for malware detection using convolutional neural networks. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2224–2231. IEEE, 2017.

[46] Hiran V Nath and Babu 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*, pages 440–450. Springer, 2014.

[47] A Ojugo and AO Eboka. Signature-based malware detection using approximate boyer moore string matching algorithm. *International Journal of Mathematical Sciences and Computing*, 5(3):49–62, 2019.

[48] Samuel Saxe, Konstantin Berlin, and S V N Vishwanathan. A deep learning approach for malware detection using recurrent neural networks. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 11–20. IEEE, 2015.

[49] James Scott. Signature based malware detection is dead. *Institute for Critical Infrastructure Technology*, 2017.

[50] Paulo F Sequeira, Pedro M Moreira, Henrique D Santos, Ricardo Fernandes, and Nuno Neves. Unsupervised deep learning for malware classification using convolutional autoencoder neural networks. *arXiv preprint arXiv:1904.03670*, 2019.

[51] Fengjun Shang, Yalin Li, Xiaolin Deng, and Dexiang He. Android malware detection method based on naive bayes and permission correlation algorithm. *Cluster Computing*, 21(1):955–966, 2018.

[52] M. S. Siddiqui, B. Sikdar, and S. Chakraborty. Deep learning for malware detection using spectral graph convolutional networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[53] SonicWall. Sonicwall 2022 cyber threat report, 2022. Accessed: April 19, 2023.

[54] Alireza Souri and Rahil Hosseini. A state-of-the-art survey of malware detection approaches using data mining techniques. *Human-centric Computing and Information Sciences*, 8(1):1–22, 2018.

[55] John W Stokes, Yuhang Li, and Sriram Chellappan. Hierarchical hidden markov models for malware detection. In *2013 10th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, pages 113–119. IEEE, 2013.

[56] Juan González Vila, Nur Azman Abu Anuar, José Sanz Soto, and Miguel Valero. Behavioral graph-based malware detection. *Expert Systems with Applications*, 41(9):4259–4274, 2014.

[57] R Vinayakumar, Raksha Shetty, and M Shridhar Bhat. Malware detection using machine learning: An introduction and review. *Journal of Information Security and Applications*, 50:102419, 2020.

[58] P Vinod, R Jaipur, V Laxmi, and M Gaur. Survey on malware detection methods. In *Proceedings of the 3rd Hackers' Workshop on computer and internet security (IITKHACK'09)*, pages 74–79, 2009.

[59] Shunxiang Wang, Hao Jiang, and Lizhong Zhu. Effective and efficient malware detection based on supervised learning techniques. In *2015 9th International Conference on Frontier of Computer Science and Technology*, pages 58–64. IEEE, 2015.

[60] Shailendra Yadav, Sarvesh Kumar Tanwar, Neeraj Kumar, and Surbhi Tyagi. Malware classification using disassembly-based features and machine learning. *Journal of Ambient Intelligence and Humanized Computing*, 10(9):3381–3392, 2019.

[61] Sumit Yadav, Adam Wierzbicki, and Chen Li. A graph-based approach for malware detection using network traffic analysis. *Journal of Network and Computer Applications*, 116:110–122, 2018.

[62] Rizwan Yasin, Umar Saeed, Muhammad Sher, and Zia Ur Rehman. An efficient heuristic-based malware detection system for windows executables. In *2018 14th International Conference on Emerging Technologies (ICET)*, pages 1–6. IEEE, 2018.

[63] Yanfang Ye, Lingwei Chen, Shifu Hou, William Hardy, and Xin Li. Deepam: a heterogeneous deep learning framework for intelligent malware detection. *Knowledge and Information Systems*, 54:265–285, 2018.

[64] Yanfang Ye, Shifu Hou, Lingwei Chen, Jingwei Lei, Wenqiang Wan, Jiabin Wang, Qi Xiong, and Fudong Shao. Out-of-sample node representation learning for heterogeneous graph in real-time android malware detection. In *28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[65] Heng Yin, Dawn Song, and Manuel Egele. Panorama: Capturing system-wide information flow for malware detection and analysis. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS)*, 2014.

[66] Xiaojie Zhang, Haihui Mao, Xinyu Liu, and Feng Chen. Generative adversarial networks for malware variant generation and detection. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9. IEEE, 2019.

[67] Yulei Zhang and Xiaokang Yang. Malware detection based on static features extraction of api calls. *Multimedia Tools and Applications*, 75(19):12001–12016, 2016.

[68] Yajin Zhou, Xuxian Jiang, Fang Zhang, and Peng Ning. A machine learning approach to android malware detection. In *Proceedings of the 5th International Conference on Trust and Trustworthy Computing*, pages 138–153. Springer, 2012.