Graph Representation Learning for Complex Security Problems

Alfredo Cuzzocrea [1,2,*], Miguel Quebrado [3], Abderraouf Hafsaoui [1] and Edoardo Serra [3]

[1] *iDEA Lab, University of Calabria, Rende, Italy*
[2] *Department of Computer Science, University of Paris City, Paris, France*
[3] *Computer Science Department, Boise State University, Boise, ID, USA*

Abstract

The polymorphic nature of malware makes it challenging to identify, especially when employing *hash-based detection approaches*, making malware detection an intriguing study topic. In contrast to image-based methods, a *graph-based method* was employed in this study to extract control flow graphs from Android APK binaries. We use a method that combines *XGBoost*, a common machine learning model, with *Inferential SIR-GN* for Graph representation, a novel graph representation learning method that preserves graph structural similarities, to handle the resulting graph. The method is then used on MALNET, an open cybersecurity database containing 1,262,024 million Android APK binary files in total, with 47 kinds and 696 families. The experimental findings show that, in terms of detection accuracy, our graph-based technique surpasses the image-based method.

Keywords

Structural Graph Representation Learning, Malware Polymorphism

1. Introduction

Malicious cyber activity's economic effects on the US economy are difficult to estimate, however, these assaults cost the economy between $57 billion to $109 billion in 2016 [1]. *Hackers* use modern tactics, technologies, and polymorphic approaches to infiltrate networks in today's data-driven corporate sector. *Cyberattacks* are mostly sophisticated and directed towards governments and huge corporations in order to disrupt main services and steal copyrights [2].

This type of attacks is achievable and successful due to malware apps. Detecting such applications can be a challenging process, however, there exist two popular methods for malware analysis: *static code analysis* and *dynamic code analysis*. The static analysis searches for malicious patterns by disassembling the code and studying the executable's control flow without executing the code. On the other hand, the code is executed virtually in the case of dynamic analysis, this approach is *behavior-based*, and therefore the key methods may be discovered.

Although static analysis provides comprehensive coverage, it still suffers occasionally from *code obfuscation*. Before analysis, the executable must be unpacked and encrypted, but regardless of that, the analysis still can be vulnerable to difficulties of intractable complexity. On the other hand, the executable does not need to be neither unpacked nor encrypted in the case of the dynamic analysis. However, it is unfortunate that dynamic analysis can still be *time-consuming* and *resource-intensive*, as mentioned in [3]. Furthermore, due to the fact that the environment does not meet the triggering criteria. Several malicious activities may be undetected [3]. The industry has moved to *image-based* malware presentations in the case of Windows and Android malware because they are faster to build, do not need feature engineering, and are resistant to multiple *standard obfuscation strategies* (e.g., encryption [3]).

On the other hand, static analysis is successful in the context of Android OS, and *graph control flow* is extractable. Furthermore, similar to the image, once the graphs are created, they do not require any

* This research has been made in the context of the Excellence Chair in Big Data Management and Analytics at University of Paris City, Paris, France
SEBD 2023: 31st Symposium on Advanced Database Systems, July 2-5, 2023, Galzignano Terme, Padua, Italy
EMAIL: alfredo.cuzzocrea@unical.it (A. Cuzzocrea); miguelquebrado@u.boisestate.edu (M. Quebrado); ahafsaoui.idealab.unical@gmail.com (A. Hafsaoui); edoardoserra@boisestate.edu (E. Serra)
ORCID: 0000-0002-7104-6415 (A. Cuzzocrea); 0000-0002-2364-172X (A. Hafsaoui); 0000-0003-0689-5063 (E. Serra)

additional engineering processes because the well-established field of graph representation learning develops the feature representing the graph automatically.

Graph representation learning techniques are promoting the advancement of representation learning as they have been adopted by various scientific domains. Structured data is encoded by these techniques into *low-dimensional space* for several essential downstream tasks (for example, toxic molecule detection, community clustering, and malware detection) [4].

Graph representation learning approaches are classified as methods that preserve *node connectivity information* and methods that preserve *node structure information*. Despite, the existence of many works that concentrate on maintaining node connections, just few works aim at maintaining node structures. For many real-world applications, properly encoding node structure information is critical since it has been proven that this information may be used to solve numerous problems where connectivity-based approaches fail [5]. Malware analysis using control flow graph extraction is another area where the structural structure of the graph helps to identify malicious from benign activity.

In this study, a graph representation learning method is used namely the *Inferential Structural Iterative Representation Learning Approach for Graph Nodes (Inferential SIR-GN)*. Which is a graph representation learning approach where theoretically the conservation of graph structural similarities is ensured. In order to identify malware, classify *Android APK types*, and classify *Android APK families*, our technique, Inferential SIR-GN, is combined with XGBoost (i.e., a common classification machine learning model).

This technique is then applied to *MALNET-TINY*, which is a subset of *MALNET* a public dataset of 1,262,024 million Android APK files divided into 47 kinds and 696 families. MALNET is one of the finest publicly accessible repositories since it is larger and includes more types than others like [1,3,6,7,8,9,10,11,12,13,14].

Our investigation on MALNET-TINY demonstrates that for malware classification and detection, Inferential SIR-GN is frequently superior or at worst similar to *ResNet* (i.e., a neural network for recognizing images) [1]. Furthermore, a strategy for obtaining *malware's obfuscated polymorphic evolution* is defined using representations from malware's inferential SIR-GN and benign Android APKs. In our investigation, we show the value of including representations of obfuscated polymorphic malware evolutions in the XGBoost training when the train and test split is done based on the APK generation date.

2. Contextual Knowledge

MALNET is the biggest cybersecurity dataset ever released; it comprises 1,262,024 Android APK files comprising 47 malware kinds and 696 malware families. The imbalance ratios for both types and families are 7,827 × and 16,901 ×, respectively. We will be dealing with MALNET-TINY in this work; as displayed in Figure. 2, both type and family have distributions with imbalance ratios of 154 × and 908 ×. MALNET-TINY involves 61,201 training, 8,743 validation, and 17,486 tests of Android APK files for type-level classification experiments by deleting MALNET's four major kinds. MALNET-TINY's purpose is to enable users to quickly prototype new concepts by requiring only a fraction of the time required to train a new model [1]. MALNET-TINY is compared to the ideal model discovered by Freitas, Duggal, and Chau [1], *ResNet18* was trained from scratch on grayscale images using *cross-entropy loss* and class re-weighting, which attained a macro-F1 score of 0.651, a macro-precision of 0.672, and a macro-recall of 0.646 [1]. Evolution prediction tests will be conducted using MALNET-TINY and *VirusTotal*. VirusTotal not only informs you whether a particular antivirus solution identified a supplied file as dangerous, but it also provides each engine's detection label (for example, I-Worm.Allaple.gen) [44]. VirusTotal routinely updates malware signatures as they are supplied by antivirus firms; this guarantees that the service utilizes the most recent signature sets, which is vital for malware scan dates.

Scale. MALNET-TINY includes 87,430 Android APK files from 43 malware categories and 246 malware families. MALNET-TINY takes up more than 35 GB of disk space in edge list format. Descriptive data on the number of nodes, edges, and average degree of MALNET-TINY are provided in Figure. 4.

Hierarchy*.** Using the Euphony [45] categorization structure, an Android APK contains function call graphs that are assigned a broad type (e.g., Fakeapp) and specific family label (e.g., Artemis), for more details, see Figure. 3. [45] defines four fields: ***type (the sort of threat, i.e., Trojan, worm, etc.), ***platform*** (the operating system that the threat is meant to run on, i.e., Windows, Android, etc.), ***family*** (the group of threats with which it is related in terms of behavior), and ***information*** (extra description of this threat, including its variant). In this study, we will concentrate on type and family.

***Diversity*.** MALNET-TINY has 43 types, 246 families, and graphs with 17,588 nodes, 40,105 edges, and 2 degrees on average. Figure. 2 depicts a type and family distribution with ratios of 154 × and 908 ×. The graphs have a long-tailed distribution, which makes classification challenging because ignoring unusual events is likely to result in high-severity mistakes during testing. When there are disparities in the scales of the input variables, the difficulty of the problem being represented increases. The distribution of hundreds or thousands of types and families presented in Figure. 2 might result in a model that learns huge weight values, which is an undesirable behavior.

***Imbalance*.** Long-tail distribution models tend to favor the majority class, resulting in poor generalization performance for uncommon classes. Traditionally, the class imbalance is resolved by resampling the data (*under-sampling*, *over-sampling*) [4] [46]. Under-sampling is commonly utilized in the field of class-imbalance learning.

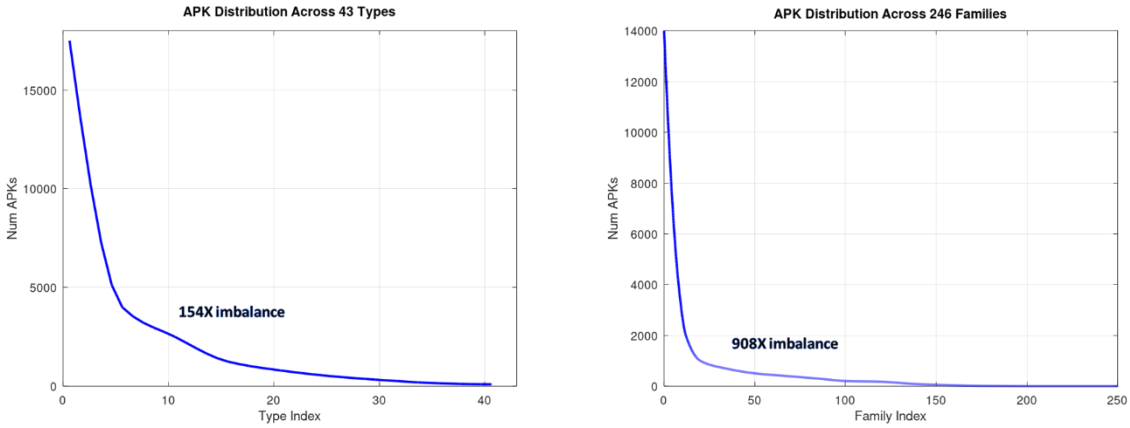


Figure 2: Android APK File Type and Family Distributions in MALNET-TINY

The fundamental shortcoming of most existing under-sampling algorithms is that their data sampling strategies are heuristic-based and unaffected by the classifier and evaluation measure utilized. As a result, during data sampling, they may ignore informative occurrences for the classifier [46]. *Random minority over-sampling (ROS)* and *random majority under-sampling (RUS)* are the two most frequent preprocessing approaches [47]. In ROS, instances of the minority class are replicated at random. In RUS, occurrences of the majority class are eliminated from the dataset at random. Kubat and Matwin [48] presented *one-sided selection (OSS)* as one of the first attempts to improve the performance of random resampling.

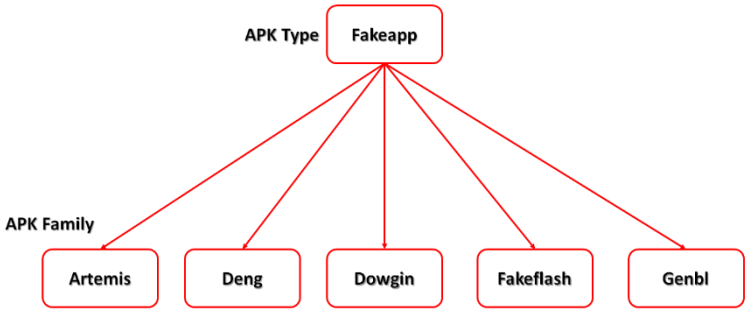


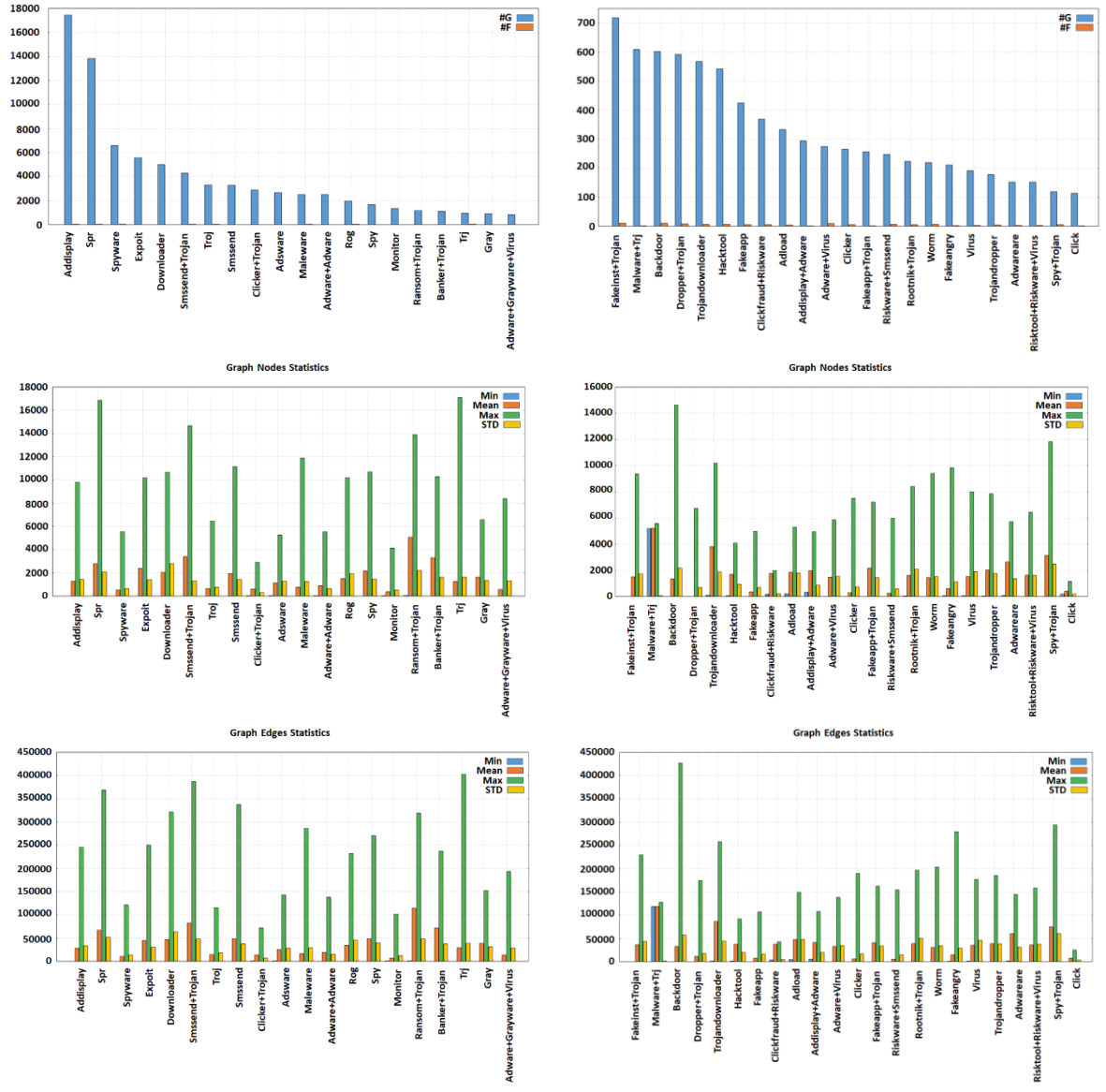
Figure 3: Graph Type "Fakeapp" and Its 5 Families

We will not use one-sided selection, which aims to intelligently under-sample the majority class by deleting majority class cases that are considered noise. In order to anticipate malware development, we will combine malware minority samples with "*benign*" ones.

For the case of the MALNET-TINY dataset the Figure. 4 displays the MALNET-TINY's graph statistics. Such that for every type included in the dataset, the Figure presents some statistics:

- $\#G$: Number of graphs included in the following type.
- $\#F$: the number of families included in the following type.
- *Nodes*: these represent the *Min*, *Max*, *Mean*, and *STD* number of the node.
- *Edges*: these model the *Min*, *Max*, *Mean*, and *STD* number of edges involved in graphs of this malware type.
- *Avg. Degrees*: finally, this metric provides information about the *Min*, *Max*, *Mean*, and *STD* average degrees of the node.

As presented in the Figure. 4, the MALNET-TINY dataset contains over $80K$ software images across a hierarchy of 43 types. We can understand that this dataset is a reduction of the original MALNET dataset, this reduction is produced to allow researchers to rapidly prototype new ideas since it requires only a fraction of the time needed to train a new model.



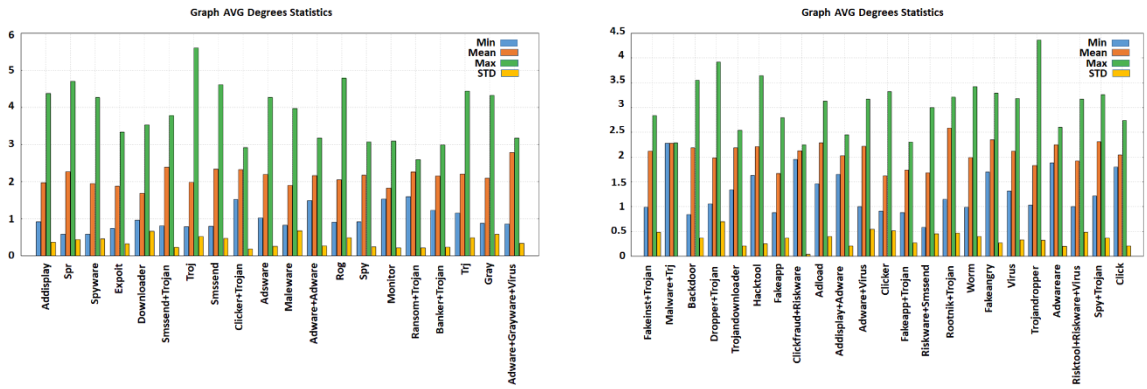


Figure 4: MALNET-TINY: Graph Statistics.

3. The Proposed Graph-Representation-Learning Framework

An approach for developing a malware classifier that is resistant to malware polymorphism is proposed. The process involves four parts:

- Extraction of the structural vectorial representation for each node in a graph describing an Android application using Inferential SIR-GN.
- Generation of the *structural pseudo-adjacency matrix* by using the vectorial representations of all the nodes of a certain graph representing an Android application. The graph is represented by the structural pseudo-adjacency matrix, followed by the Android app.
- To build a potentially polymorphic variant of the virus, we combine the malware's structural pseudo-adjacency matrix with the matrix of the benign Android app.
- Train a *random forest* algorithm to identify and categorize malware using the representations of the Android applications (*benign* and *malware*) and the representations of the hypothetical polymorphic form of the malware.

The Inferential SIR-GN, the structural pseudo-adjacency matrix, and the matrix combination for possible polymorphic combinations are all described below.

3.1. Inferential SIR-GN: Overview and Methods

Layne and Serra [49] provide a description of the Inferential SIR-GN technique, which is used to extract node representations from directed graphs. The model is based on the SIR-GN approach, which was initially published in [50], in which a node's representation is iteratively updated by characterizing and then aggregating its neighbors. At each iteration, the size of a node's representation is equal to a *user-specified hyperparameter* nr. The current node description (which starts as the node degree) is grouped into nr KMeans clusters to create node descriptions. At each iteration, the representation is normalized before the clustering step, and the distance from each *cluster centroid* is translated into a probability of the node's membership in each cluster. Since the node's structural description has been modified, its neighbors are aggregated into its description by summing all neighbors' probability of membership in each cluster. The final node representation is equal to the predicted number of neighbors in each cluster for that node. Each iteration corresponds to a greater depth of exploration, with N iterations producing a node description that incorporates a node's *N-hop neighborhood structure*.

The first difference between inferential SIR-GN and the standard model is that at the conclusion of each iteration, each node's structural description is concatenated into a bigger representation that reflects the evolution of the structural information via deeper neighborhood exploration. A *Principle Component Analysis (PCA)* is employed after the final iteration to prevent information erosion as the representation size rises. The final representation is reduced to a hyperparameter-specified size. A node's initial representation in a directed graph begins with two vectors of size nr, one providing the node's in-degree and the other holding its out-degree. Before clustering, these two vectors are concatenated. Clustering of this bigger node vector is conducted at each iteration, then by aggregation of the neighbors. In the case of directed data, aggregation is performed independently for a node's in-

neighbors and out-neighbors into two intermediate vectors, which are then concatenated together for the following iteration. The proposed model's inferential capacity is achieved by pre-training the KMeans and scalers. For each iteration, we employ a new KMeans and Scaler for each depth of investigation, in addition to the PCA model that will be used to build the final node representation. Each model is pre-trained on random graphs and saved for later use in inference. At inference time, we employ the pre-trained models to perform repeated normalizing, clustering, and aggregation, and the PCA fit during training is used to construct the final node representations. This reduces inference time significantly, and the same pre-trained model may be utilized on a range of different data sources. Layne and Serra prove that, along with a full method and description of the model's temporal complexity.

3.2. A Novel Concept of Adjacency Matrix

A process for creating a unique graph representation approach is provided by [49] based on the vectorial representation of SIR-GN. Such approaches identify fixed-number groupings of nodes. Each group comprises nodes with vectorial representations that are comparable. Given this collection of groups, a structural pseudo-adjacency matrix based on the groups is generated, which provides the vectorial representation of the network once flattened. The vectorial representations of the two graphs are thus comparable if the calculation of the node representations and the specification of the node groups for the structural pseudo-adjacency matrices for the two graphs are the same. This characteristic is guaranteed by this technique since inferential SIR-GN [49] is a process that can make inferences and is pre-trained on a certain family of directed random graphs. The graph representation is invariant since the groups are formed based on structural similarities between the nodes.

These node representations are especially utilized to train a final scalar and KMeans model that clusters the complete graph data at inference time. This final KMeans is fitted using the concatenated iterative node representations compressed by PCA, as opposed to the incremental KMeans, which only observes the node representation or aggregation for the current level of depth being examined. During inference, the nodes of the target graph are embedded as stated above, then grouped again using the KMeans pre-trained on the entire graph data. As noted previously, the distances to the cluster centroids are converted into probabilities of cluster membership. Nevertheless, the aggregation approach for graph representation differs significantly from that for nodes. Graph representations are frequently produced by summing or mean-pooling node representations. A novel approach for node pooling is provided by Layne and Serra [49], which generates a *structural pseudo-adjacency matrix* of dimension $ams \times ams$, where the matrix is the sum of each node vector multiplied by the transpose of each of its neighbors. Unlike traditional adjacency matrices, this produces a matrix that is not unique to a certain network topology but also agnostic to node ordering. A set of characteristics is produced by the linearized matrix that may be used in subsequent graph classification tasks.

3.3. Malware Polymorphic Generation Analysis: New Approach

We build a process to construct a polymorphic variant of existing malware utilizing the structural pseudo-adjacency matrix that represents each Android application. $SPAM(n)$ represents the structural pseudo-adjacency matrix of the network of an Android application n. For each Android virus x, the process looks among the benign apps for the application m that has $SPAM(m)$ closest in terms of Euclidean distance to $SPAM(m)$. The *k-nearest neighbor technique (KNN)* is used to compute the closest benign application quickly. Given the Android malware a and the closet innocuous application m, the following weighted mean of the two representations yields a polymorphic representation pr_n of n:

$$pr_n = 0.8 \cdot SPAM(n) + 0.2 \cdot SPAM(m) \quad (1)$$

Weights are used to subtly alter the virus representation $SPAM(n)$. Then, we develop a polymorphic representation for each malware, and all of the polymorphic representations are employed in the training of the classification model, in this case, a random forest, to make the classification model resistant to polymorphic changes in malware applications.

The latter one is among the most relevant innovations of our proposed research, positioning itself as a noticeable contribution to the actual research. Polymorphic analysis, in fact, is critical for malware detection.

4. Experimental Evaluation and Analysis

4.1. Setup

MALNET-TINY 5k is divided into 3,500 training, 500 validation, and 1,000 graphs for type-level classification tests. MALNET-TINY is composed of 61,201 training graphs, 8,743 validation graphs, and 17,486 graphs. For type-level classification experiments and evolution prediction classification, we split these datasets. In this investigation, a comparison is made against Freitas, Duggal, and Chau [1], who used a ResNet18 model trained from scratch on grayscale pictures using cross-entropy loss and class reweighting and achieved a macro-F1 score of 0.651, a macro-precision of 0.672, and a macro-recall of 0.646 [1]. MALNET-TINY and MALNET-TINY 5k were also analyzed by carrying out *type-level classification* studies on various data splits. Each dataset is changed using the SIR-GN approach, which encodes node structure, and with multiple iterations, it develops rich structural representations by using node clustering and node neighborhood interactions [5]. The final result of SIR-GN is a structural representation vector that is input into an XGBoost classifier. In addition to the *macro-F1 score*, we give other performance indicators like *accuracy* and *recall*.

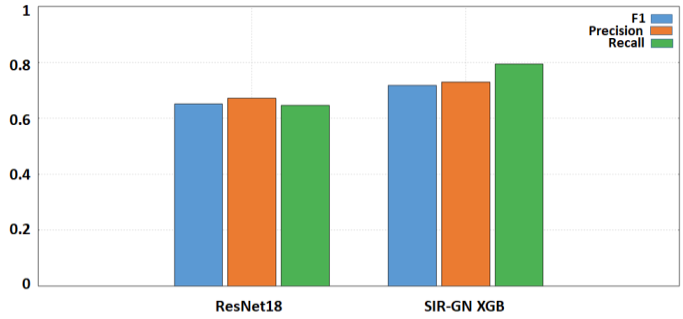


Figure 5: SIR-GN Against ResNet18 Comparison over MALNET-TINY 61k

4.2. Results

The results for the SIR-GN graph representation approach [49] described in Section III are provided, as well as results for MALNET-TINY 5k obtained in https://mal-net.org/ and also for MALNET-TINY which is divided into 61,201 training, 8,743 validation, and 17,486 test graphs [1]. Our investigations are performed in Python3 on an Intel (R) Core(TM) $i7-7700HQ\ CPU$ @ 2.80 GHz.

The following datasets are used:

- **MALNET-TINY 5k** each of dataset's kinds (Addisplay, Adware, Benign, Downloader, and Trojan) has 1000 graphs.
- **MALNET-TINY 61K** For type-level classification tests, there are 61,201 training, 8,743 validation, and 17,486 test graphs.
- **MALNET-TINY 81K** For type level classification experiments and evolution prediction, there are 81,201 training, 8,743 validation, and 27,486 test graphs.

The results for the MALNET-TINY dataset using macro-F1, macro-precision, and macro-recall on random splits of 61,201 training graphs, 8,743 validation graphs, and 17,486 test graphs are shown in Figure. 5. We compared our results to those of [1] who had a macro-F1 score of 0.651, a macro-precision of 0.672, and a macro-recall of 0.646, whereas we obtain a macro-F1 score of 0.718, a macro-precision of 0.729, and a macro-recall of 0.794. Using the SIR-GN algorithm increases the classifier's performance.

Figure. 6 displays the findings for the MALNET-TINY 5k dataset with macro-F1, macro-precision, and macro-recall performed on random splits of 3,500 training graphs, 500 validation graphs, and 1000 test graphs. We achieved a macro-F1 score of 0.916, a macro-precision of 0.917, and a macro-recall of 0.915 for the random split.

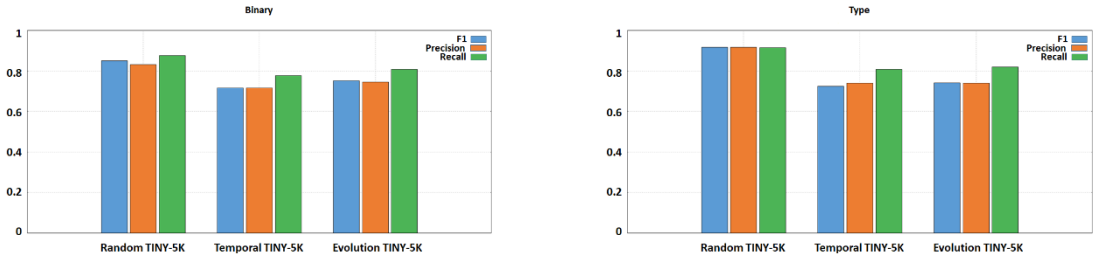


Figure 6: SIR-GN Performance over MALNET-TINY 5k with Different Splitting Policies

As reported in Figure. 7, accuracy scores are presented for numerous graph-based approaches [51], with the Inferential SIR-GN approach coming out on top with 0.92 accuracies. This Figure also shows results for a temporal split in which the training dates from 2012 to 2019, while the test dates ranged from 2020 to 2021. A macro-F1 score of 0.725, a macro-precision of 0.739, and a macro-recall of 0.807 were obtained for the temporal split and type categorization. When we use evolution prediction, we achieve a 0.741 vs 0.725 performance improvement for a macro-F1 score.

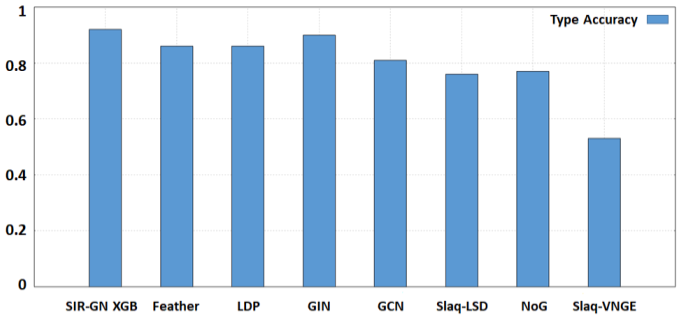


Figure 7: Comparison of Inferential SIR-GN Against Several Methods [51] over MALNET-TINY 5k

In the experiments presented in Figure. 8, we compare MALNET-TINY against benign malware types to create a score based on a random split. The macro-F1 score, macro-precision, and macro-recall for random and temporal splits are presented in this Figure. The Figure also displays results for a temporal split in which the training dates ranged from 2012 to 2019, while the test dates ranged from 2020 to 2021. We obtained a macro-F1 score of 0.725, a macro-precision of 0.739, and a macro-recall of 0.807 for the temporal split and type categorization. When we use evolution prediction, we achieve a 0.741 vs 0.725 performance improvement for a macro-F1 score.

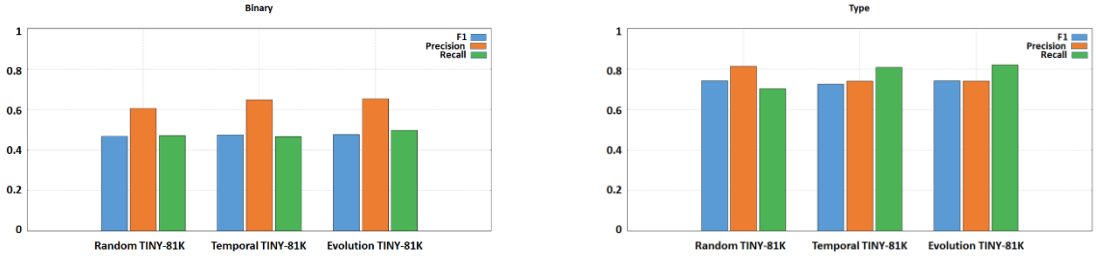


Figure 8: SIR-GN Performance over MALNET-TINY 81k with Different Splitting Policies

5. Conclusions and Future Work

In this study, MALNET and emphasized graph representation learning approaches that have evolved across various scientific domains were presented, these methods are driving the evolution of representation learning techniques. For several essential downstream tasks, graph representation learning approaches compress structured information into low-dimensional space. We suggested an approach for developing a malware classifier that is resistant to malware polymorphism. Given a graph representing an Android application, Inferential SIR-GN was used to extract the structural vectorial representation for each node within the network. In this study, we trained a model to recognize and classify malware using representations of Android applications (both benign and malicious) and possibly polymorphic versions of the malware.

In the investigation, we go on with the experimental settings and the empirical results. SIR-GN graph representation technique results are presented. According to our findings, Malware applications are *harmful* and cause significant damage. As a result, malware identification and categorization are critical for its mitigation. Transforming binary executables into pictures is a practical and accurate method for using image neural networks to identify and categorize malware. In contrast to the Image-based technique, we proposed in this paper a procedure based on control flow graph, structural graph representation learning, and XGBoost. In terms of malware classification, such a process was evaluated on MALNET and outperformed the image-based method utilizing the neural network ResNet. Moreover, we created a process to generate a polymorphic version of an existing malware program utilizing the structural pseudo-adjacency matrix representing each Android application in order to improve the training phase and classification performances.

Future work is mainly oriented to embed performance in our framework, as required by modern big data trends (e.g., [57-60]).

6. References

[1] S. Freitas, R. Duggal, D.H. Chau, "MALNET: A Large-Scale Cybersecurity Image Database of Malicious Software", in: *31st ACM International Conference on Information & Knowledge Management*, 2022, pp. 3948-3952

[2] S. Freitas, A. Wicker, D.H. Chau, J. Neil, "D2M: Dynamic Defense and Modeling of Adversarial Movement in Networks", in: *2020 SIAM International Conference on Data Mining*, 2020, pp. 541-549

[3] L. Nataraj, S. Karthikeyan, G. Jacob, B.S. Manjunath, "Malware Images: Visualization and Automatic Classification", in: *8th International Symposium on Visualization for Cyber Security*, 2011, art. 4

[4] S. Freitas, Y. Dong, J. Neil, D.H. Chau, "A Large-Scale Database for Graph Representation Learning", *The Neural Information Processing Systems Track on Datasets and Benchmarks 1*, 2021

[5] M. Joaristi, E. Serra, "SIR-GN: A Fast Structural Iterative Representation Learning Approach for Graph Nodes", *ACM Transactions on Knowledge Discovery from Data 15.6* (2021) 100

[6] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, M. Ahmadi, "Microsoft Malware Classification Challenge", *CoRR abs/1802.10135*, 2018

[7] A.S. Bozkir, A.O. Cankaya, M. Aydos, "Utilization and Comparison of Convolutional Neural Networks in Malware Recognition", in: *27th Signal Processing and Communications Applications Conference*, 2019, pp. 1-4

[8] T.M. Mohammed, L. Nataraj, S. Chikkagoudar, S. Chandrasekaran, B.S. Manjunath, "Malware Detection Using Frequency Domain-Based Image Visualization and Deep Learning", in: *54th Hawaii International Conference on System Sciences*, 2021, pp. 1-10

[9] L. Chen, R. Sahita, J. Parikh, M. Marino, "Stamina: Scalable Deep Learning Approach for Malware Classification". *Intel White Paper 1.3* (2020)

[10] J. Gennissen, J. Blasco, "GAMUT: Sifting Through Images to Detect Android Malware". *Bachelor thesis, Royal Holloway University, London, UK*, 2017

[11] K. Kancherla, S. Mukkamala, “Image Visualization Based Malware Detection”, in: *2013 IEEE Symposium on Computational Intelligence in Cyber Security*, 2013, pp. 40-44
[12] S. Choi, S. Jang, Y. Kim, J. Kim, “Malware Detection using Malware Image and Deep Learning”, in: *2017 International Conference on Information and Communication Technology Convergence*, 2017, pp. 1193-1195
[13] H. Zhang, X. Xiao, F. Mercaldo, S. Ni, F. Martinelli, A.K. Sangaiah, “Classification of Ransomware Families with Machine Learning Based on N-Gram of Opcodes” *Future Generation Computer Systems 90.8* (2019) 211-221
[14] J. Su, D.V. Vasconcellos, S. Prasad, D. Sgandurra, Y. Feng, K. Sakurai, “Lightweight Classification of IoT Malware Based on Image Recognition”, in: *42nd IEEE Annual Computer Software and Applications Conference 2*, 2018, pp. 664-669
[15] I. Yoo, “Visualizing Windows Executable Viruses using Self-Organizing Maps”, in: *2004 ACM Workshop on Visualization and Data Mining for Computer Security*, 2004, pp. 82-89
[16] Z. Yuan, Y. Lu, Y. Xue, “Droiddetector: Android Malware Characterization and Detection using Deep Learning”. *Tsinghua Science and Technology 21.1* (2016) 114-123
[17] H. Gascon, F. Yamaguchi, D. Arp, K. Rieck, “Structural Detection of Android Malware using Embedded Call Graphs”, in: *2013 ACM Workshop on Artificial Intelligence and Security*, 2013, pp. 45-54
[18] S. Ranveer, S. Hiray, “Comparative Analysis of Feature Extraction Methods of Malware Detection”. *International Journal of Computer Applications 120.5* (2015) 1-7
[19] M. Nunes, “Dynamic Malware Analysis Kernel and User-Level Calls” 2018
[20] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, J. Leskovec, “Open Graph Benchmark: Datasets for Machine Learning on Graphs”, *Advances in Neural Information Processing Systems 33* (2020) 22118-22133
[21] X. Yan, H. Cheng, J. Han, P.S. Yu, “Mining Significant Graph Patterns by Leap Search”, in: *ACM SIGMOD International Conference on Management of Data*, 2008, pp. 433-444
[22] X. Kong, P.S. Yu, “Multi-Label Feature Selection for Graph Classification”, in: *10th IEEE International Conference on Data Mining*, 2010, pp. 274-283
[23] H. NT, C.J. Jin, T. Murata, “Learning Graph Neural Networks with Noisy Labels”, *CoRR abs/1905.01591*, 2019
[24] A. Lusci, G. Pollastri, P. Baldi, “Deep Architectures and Deep Learning in chemoinformatics: The Prediction of Aqueous Solubility for Drug-Like Molecules”, *Journal of Chemical Information and Modeling 53.7* (2013) 1563-1575
[25] K. Riesen, H. Bunke, “IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning”, in: *Joint International Workshops on Statistical Techniques in Pattern Recognition and Structural and Syntactic Pattern Recognition. Springer 5342*, 2008, pp. 287-297
[26] K.M. Borgwardt, C.S. Ong, S. Schönauer, S.V.N. Vishwanathan, A.J. Smola, H.P. Kriegel, “Protein Function Prediction via Graph Kernels”, *Bioinformatics 21(1)* (2005) 47-56
[27] X. Yue, Z. Wang, J. Huang, S. Parthasarathy, S. Moosavinasab, Y. Huang, S.M. Lin, W. Zhang, P. Zhang, H. Sun, “Graph Embedding on Biomedical Networks: Methods, Applications and Evaluations”, *Bioinformatics 36(4)* (2020) 1241-1251
[28] B. Rozemberczki, O. Kiss, R. Sarkar, “Karate Club: An API Oriented Open-Source Python Framework for Unsupervised Learning on Graphs”, in: *29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 3125-3132
[29] L. Nataraj, V. Yegneswaran, P.A. Porras, J. Zhang, “A Comparative Assessment of Malware Classification using Binary Texture Analysis and Dynamic Analysis”, in: *4th ACM Workshop on Security and Artificial Intelligence*, 2011, pp. 21-30
[30] M. Ceci, A. Cuzzocrea, D. Malerba, “Supporting Roll-Up and Drill-Down Operations over OLAP Data Cubes with Continuous Dimensions via Density-Based Hierarchical Clustering”, in: 1*9th Italian Symposium on Advanced Database Systems*, 2011, pp. 57-65
[31] E. Serra, M. Joaristi, A. Cuzzocrea, “Large-Scale Sparse Structural Node Representation”, in: *2020 IEEE International Conference on Big Data*, 2020, pp. 5247-5253
[32] P. Braun, A. Cuzzocrea, T.D. Keding, C.K. Leung, A.G.M. Padzor, D. Sayson, “Game Data Mining: Clustering and Visualization of Online Game Data in Cyber-Physical Worlds”, *Procedia Computer Science 112* (2017) 2259-2268

[33] A. Guzzo, D. Sacca, E. Serra, "An Effective Approach to Inverse Frequent Set Mining", in: *9th IEEE International Conference on Data Mining*, 2009, pp. 806-811
[34] K.J. Morris, S.D. Egan, J.L. Linsangan, C.K. Leung, A. Cuzzocrea, C.S.H. Hoi, "Token-Based Adaptive Time-Series Prediction by Ensembling Linear and Non-Linear Estimators: a Machine Learning Approach for Predictive Analytics on Big Stock Data", in: *17th IEEE International Conference on Machine Learning and Applications*, 2018, pp. 1486-1491
[35] E. Serra, V. Subrahmanian, "A Survey of Quantitative Models of Terror Group Behavior and an Analysis of Strategic Disclosure of Behavioral Models", *IEEE Transactions on Computational Social Systems 1.1* (2014) 66-88
[36] L. Bellatreche, A. Cuzzocrea, S. Benkrid, "F&A: A Methodology for Effectively and Efficiently Designing Parallel Relational Data Warehouses on Heterogenous Database Clusters", in: *International Conference on Data Warehousing and Knowledge Discovery*, 2010, pp. 89-104
[37] O. Korzh, M. Joaristi, E. Serra, "Convolutional Neural Network Ensemble Fine-Tuning for Extended Transfer Learning". in: *International Congress on Big Data*, 2018, pp. 110-123
[38] S. Ahn, S.V. Couture, A. Cuzzocrea, K. Dam, G.M. Grasso, C.K. Leung, K.L. McCormick, B.H. Wodi, "A Fuzzy Logic Based Machine Learning Tool for Supporting Big Data Business Analytics in Complex Artificial Intelligence Environments", in: *2019 IEEE International Conference on Fuzzy Systems*, 2019, pp. 1-6
[39] E. Serra, A. Sharma, M. Joaristi, O. Korzh, "Unknown Landscape Identification with CNN Transfer Learning", in: *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2018, pp. 813-820
[40] E. Serra, A. Shrestha, F. Spezzano, A.C. Squicciarini, "DeepTrust: An Automatic Framework to Detect Trustworthy Users in Opinion-Based Systems", in: *10th ACM Conference on Data and Application Security and Privacy*, 2020, pp. 29-38
[41] M. Joaristi, E. Serra, F. Spezzano, "Inferring Bad Entities Through the Panama Papers Network", in: *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2018, pp. 767-773
[42] M. Joaristi, E. Serra, F. Spezzano, "Detecting Suspicious Entities in Offshore Leaks Networks". *Social Network Analysis and Mining 9.1* (2019) 62
[43] G. Sood, "virustotal: R Client for the virustotal API". *R package version 0.2.1*, 2017
[44] M. Hurier, G. Suarez-Tangil, S.K. Dash, T.F. Bissyandé, Y. Le Traon, J. Klein, L. Cavallaro, "Euphony: Harmonious Unification of Cacophonous Anti-Virus Vendor Labels for Android Malware", in: *14th IEEE/ACM International Conference on Mining Software Repositories*, 2017 pp. 425-435
[45] M. Peng, Q. Zhang, X. Xing, T. Gui, X. Huang, Y.G. Jiang, K. Ding, Z. Chen, "Trainable UnderSampling for Class-Imbalance Learning", in: *AAAI Conference on Artificial Intelligence 33.1*, 2019, pp. 4707-4714
[46] J. Van Hulse, T.M. Khoshgoftaar, A. Napolitano, "Experimental Perspectives on Learning from Imbalanced Data", in: *24th International Conference on Machine Learning*, 2007, pp. 935-942
[47] M. Kubat, S. Matwin, "Addressing the Curse of Imbalanced Training Sets: One-Sided Selection", in: *14th International Conference on Machine Learning*, 1997, pp. 179-186
[48] J. Layne, E. Serra, "Inferential SIR-GN: Scalable Graph Representation Learning", *CoRR abs/2111.04826*, 2021
[49] M. Joaristi, E. Serra, "SIR-GN: A Fast Structural Iterative Representation Learning Approach for Graph Nodes", *ACM Transactions on Knowledge Discovery from Data 15.6* (2021) 100
[50] S. Freitas, Y. Dong, J. Neil, D.H. Chau, "A Large Scale Database for Graph Representation Learning", in: *1st NeurIPS Datasets and Benchmarks*, 2021
[51] B. Rozemberczki, R. Sarkar, "Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models", in: *29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 1325-1334
[52] C. Cai, Y. Wang, "A Simple Yet Effective Baseline for Non-Attributed Graph Classification", *CoRR abs/1811.03508*, 2018
[53] K. Xu, W. Hu, J. Leskovec, S. Jegelka, "How Powerful are Graph Neural Networks?", in: *7th International Conference on Learning Representations*, 2019

[54] T.N. Kipf, M. Welling, “Semi-Supervised Classification with Graph Convolutional Networks”, in: *5th International Conference on Learning Representations*, 2017
[55] A. Tsitsulin, M. Munkhoeva, B. Perozzi, “Just Slaq When You Approximate: Accurate Spectral Distances for Web-Scale Graphs”, in: *2020 Web Conference*, 2020, pp. 2697-2703
[56] T.H. Schulz, P. Welke, “On the Necessity of Graph Kernel Baselines”, in: *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases - Graph Embedding and Mining Workshop*, 2019, art. 6
[57] A. Bonifati, A. Cuzzocrea, “Storing and retrieving XPath fragments in structured P2P networks”, *Data Knowl. Eng. 59.2*, (2006) pp. 247-269
[58] A. Cuzzocrea, D. Saccà, P. Serafino, “A Hierarchy-Driven Compression Technique for Advanced OLAP Visualization of Multidimensional Data Cubes”, in: *2006 DaWaK International Conference*, 2006, pp. 106-119
[59] A. Cuzzocrea, F. Furfaro, G.M. Mazzeo, D. Saccà, “A Grid Framework for Approximate Aggregate Query Answering on Summarized Sensor Network Readings”, in: *2004 OTM International Workshops*, 2004, pp. 144-153
[60] M. Kumar, “Scalable malware detection system using big data and distributed machine learning approach”, *Soft Comput. 26.8*, pp. 3987-4003 (2022)