Android Malware Identification and Polymorphic Evolution Via Graph Representation Learning

Miguel Quebrado
Computer Science dept.
Boise State University
Boise, ID, USA
miguelquebrado@u.boisestate.edu

Edoardo Serra
Computer Science dept.
Boise State University
Boise, ID, USA
edoardoserra@boisestate.edu

Alfredo Cuzzocrea
iDEA Lab
University of Calabria,
Rende, Calabria, Italy
alfredo.cuzzocrea@unical.it

***Abstract*—Developing techniques to identify malware is critical. The polymorphic nature of malware makes it difficult to detect, especially if the detection is done with Hash-based based techniques. Image-based binary representations have been shown to be more robust to popular polymorphic obfuscation techniques. In contrast to image-based techniques, in this paper, we employed a graph-based technique that extracts control flow graphs from Android APK binary. To process the resulting graph, we use a procedure combining a new graph representation learning method, called Inferential SIR-GN for Graph representation, that preserves graph structural similarities, with XGboost, which is a standard machine learning model. Then, we apply this procedure to MALNET, which is a publicly available cybersecurity database that provides image and graph-based Android APK binary representations for a total 1,262,024 million Android APK binary with 47 types and 696 families. Experimental results show that this graph-based procedure is even more accurate than the image-based approach. Moreover, this paper provides a procedure that, by leveraging Inferential SIR-GN is able to create malware polymorphic evolution representations to use during the train of the XGboost that strengthens the malware classification tasks when the train and test datasets are split temporally according to the binary creation date. This means that our procedure can predict malware polymorphic evolution.**

***Index Terms*—Obfuscation, Neural Networks, Structural Graph Representation Learning, Malware Polymorphism.**

I. Introduction

The economic costs that malicious cyber activity has on the U.S. economy can be challenging to determine, but these attacks cost the economy anywhere between $57 billion and $109 billion in 2016[1]. In a data-driven business world, hackers leverage advanced techniques, technologies, and polymorphic methods to compromise networks. Cyberattacks are often highly sophisticated, targeting governments and large-scale enterprises to interrupt critical services and steal intellectual property[2].

Malware applications are one of the main reasons why such attacks are possible and successful. Identifying malware is a difficult task, but there are two common approaches to analyzing malware static code analysis and dynamic code analysis. Static analysis works by disassembling the code and exploring the control flow of the executable to look for malicious patterns without actually running the code. Dynamic analysis involves executing the code in a virtual environment; this approach is behavior-based, so the important methods can be identified.

The static analysis offers complete coverage, but it usually suffers from code obfuscation. The executable has to be unpacked and decrypted before analysis, and even then, the analysis can be hindered by problems of intractable complexity. The dynamic analysis does not need the executable to be unpacked or decrypted. Unfortunately, as noted in [3], dynamic analysis can still be time-intensive and resource-consuming. Moreover, some malicious behaviors might be unobserved because the environment does not satisfy the triggering condition [3]. For windows and android malware, the industry has turned to image-based malware presentations as they are quick to generate, require no feature engineering, and are resilient to some common obfuscation techniques (e.g., section encryption [3]).

However, in the specific context of Android OS, static analysis is effective, and extracting the control flow graph is doable. Moreover, similar to the image, once the graphs are produced, they do not need any specific future engineering process since the well-established field of graph representation learning automatically creates the feature representing the graph.

Graph representation learning methods have emerged across many scientific fields and are driving the development of representation learning techniques. Graph representation learning techniques encode structured information into low dimensional space for a variety of important downstream tasks (e.g., toxic molecule detection, community clustering, malware detection)[4].

Graph representation learning methods are divided into methods preserving the connectivity information of the nodes and the methods preserving nodes' structural information. While there are a lot of works that focus on preserving node connectivity, only a few works focus on preserving nodes' structure. Properly encoding nodes' structural information is fundamental for many real-world applications as it has been demonstrated that this information can be leveraged to successfully solve many tasks where connectivity-based methods usually fail [5]. Malware analysis through the extraction of control flow graphs is another field where the structural pattern of the graph can distinguish malicious from benign activities.

In this paper, we leverage a graph representation learning method called Inferential Structural Iterative Representation learning approach for Graph Nodes(Inferential SIR-GN). Inferential SIR-GN is a graph representation learning method. Theoretically, it guarantees the preservation of graph structural similarities. Our method combines Inferential SIR-GN with XGbosst (which is a standard classification machine learning model) to perform malware detection,android APK type classification, and android APK family classification.

Then, we apply our method on MALNET TINY a subset of MALNET which is a public dataset containing 1,262,024 million Android APK files with 47 types and 696 families. MALNET results as one of the best repository freely available because it is larger and it has various types when compared to other such as [[1], [3], [6], [7], [8], [9], [10], [11], [12], [13],[14].

Our experiments on MALNET TINY, shows that Inferential SIR-GN for malware classification and detection is often better or, in the worst-case comparable to RESNET (an image neural network) that classify the images extracted from the Android APKs [1].

In addition, we define a procedure that combines the representations extracted from Inferential SIR-GN of malware and benign Android APK to obtain obfuscated polymorphic evolution of the malware. In our experiments, we show the benefit of adding in the training of the XGbosst the representations of the obfuscated polymorphic malware evolutions in the case where the train and test split is done temporally according to the APK creation date.

II. Related Work

The classic virus-detection techniques look for the presence of a virus-specific sequence of instructions, called a virus signature, inside the program: if the signature is found, it is highly probable that the program is infected [15]. In [10] publication, images are created to train convolutional neural networks on malware detection. With their approach feature engineering is done by the neural network after adding domain knowledge. This ensures an automatic feature space while achieving high performance. Different levels of semantics have different results, peaking at 92% accuracy by combining domain-expert knowledge with the feature engineering by the neural network [10]. Due to the open nature of Android, countless malware applications are hidden in a large number of benign apps in Android markets that seriously threaten Android security. Deep learning is a new area of machine learning research that has gained increasing attention in artificial intelligence [16]. In [16] publication, they propose to associate the features from the static analysis with features from dynamic analysis of Android apps and characterize malware using deep learning techniques. MALNETS dataset contains images that are fast to generate, require no feature engineering, and are resilient to popular obfuscation methods. MALNET offers unique opportunities to advance the frontiers of graph representation learning, Table I contains publications enabling research into imbalanced classification, explainability, and the impact of class hardness.

TABLE I: Graph based research learning areas.

Application	Dataset	Graphs	Classes
Cybersecurity	MALNET[1]	1,262,024	696
	Google Play[17]	147,950	2
	GCD[18]	1,361	2
	DMA[19]	2,000	2
Small molecule	Molpcba[20]	437,929	2
	Yeast[21]	79,601	2
	NCI1[22]	812	10
	RDT-M5K[23]	5,000	5
	Delaney[24]	2,874	2
Computer Vision	Digit[25]	3,500	10
	Fingerprint[25]	2,800	4
	COIL-RAG[25]	3,900	100
	COIL-DEL[25]	3,900	100
Bioinformatic	PROTEINS[26]	1,178	2
	CTD-DDA[27]	12,765	2
Social Network	Reddit-T[28]	203,088	2
	Twitch Ego Nets[29]	127,094	2
	Github Stargazers[29]	12,725	2
	Reddit-12K[28]	203,088	2

Comparison of MALNET properties with common graph classification datasets found in other research areas. MALNET offers over 1.2 million Android APK files averaging 17k nodes and 39k edges with a hierarchical class structure containing 47 types and 696 families. This makes MALNET the largest public database constructed to date, offering 105× more Android APK files, 44× larger graphs on average, and 63× the classes compared to several datasets in I. To put this in perspective, MALNET's smallest class contains only 113 samples of the Click graph, while 884,455 of the Adware type[4]. A majority of newly identified malware samples are packed, meaning that the binary code is obfuscated to evade signature-based detection, the predominant form of malware detection [30]. Fortunately, research has shown that image-based binary representations are resilient to common packing techniques since they typically perform a monotonic transformation of the binaries, failing to conceal common byte patterns present in the original binaries[30]. With the release of MALNET, researchers will now have access to a critical resource to develop advanced, image-based, or graph-based malware detection and classification algorithms [1]. Previous research on classic desktop malware has shown that some high-level characteristics of the code, such as function call graphs, can be used to find similarities between samples while being more robust against certain obfuscation strategies[17]. The study of graph representation learning is a critical tool in the characterization and understanding of complex interconnected systems. Currently, no large-scale database exists to accurately assess the strengths and weaknesses of these techniques [4]. Such data represents a benchmark for testing machine learning model for cybesecurity. In [31, 32, 33, 34,

35, 36, 37, 38, 39, 40, 41, 42, 43] are reported several related works in the field of machine learning.

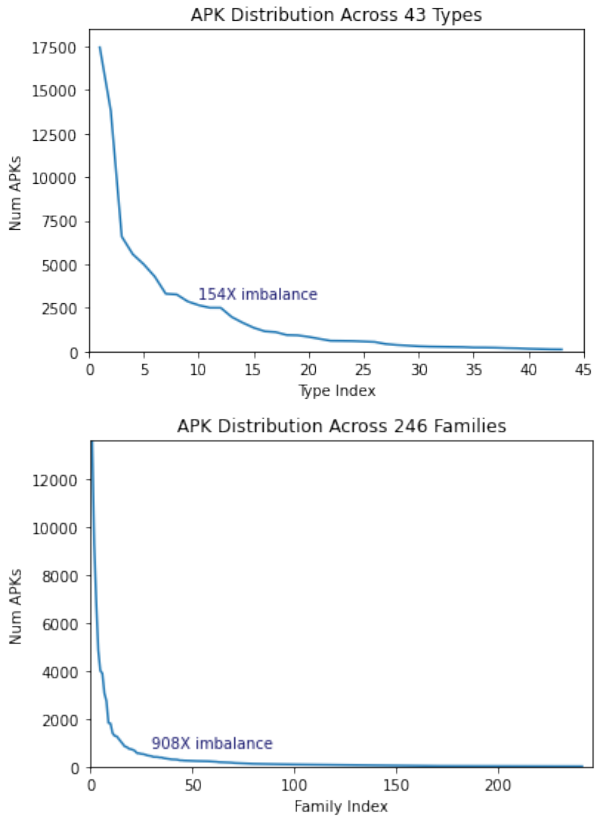

Fig. 1: MALNET-TINY contains 87,430 Android APK file across a hierarchy of 43 types and 246 families. Both type and family have distributions with imbalance ratios of 7,827× and 16,901×, respectively.

III. Background

MALNET is the largest cybersecurity dataset to date that has been released; it contains 1,262,024 Android APK files across 47 types and 696 families of malware. Both types and families have distributions with imbalance ratios of 7,827× and 16,901×, respectively. In this paper we will be working with MALNET-TINY, both type and family have distributions with imbalance ratios of 154× and 908× the distribution can be seen in Figure 1. MALNET-TINY contains 61,201 training, 8,743 validation, and 17,486 tests Android APK files, for type-level classification experiments by removing the 4 largest types in MALNET. The goal of MALNET-TINY is to enable users to rapidly prototype new ideas since it requires only a fraction of the time needed to train a new model[1]. We analyze MALNET-TINY by performing type level classification experiments against the optimal model found in Freitas, Duggal, and Chau [1], ResNet18 trained from scratch on grayscale images using cross-entropy loss and class reweighting—where the model achieves a macro-F1 score of 0.651, macro-precision of 0.672, and a macro-recall of 0.646[1]. We will also explore evolution prediction experiments with MALNET-TINY with the help of VirusTotal. VirusTotal tells you whether a given antivirus solution detected a submitted file as malicious, but also displays each engine's detection label (e.g., I-Worm.Allaple.gen) [44]. Malware signatures are updated frequently by VirusTotal as they are distributed by antivirus companies, this ensures that the service uses the latest signature sets which is important for scan dates of the malware.

We begin by analyzing 5 key properties of the MALNET-TINY (1) scale (number of graphs, average graph size, average number of nodes, average number of edges), (2) class hierarchy (3) class diversity, and (4) class imbalance.

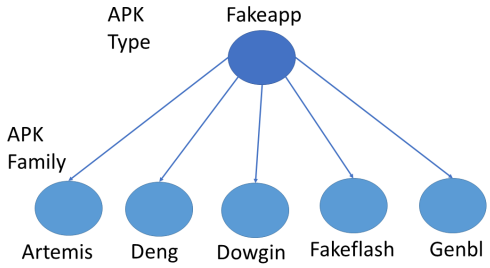

Fig. 2: Example of the graph type "fakeapp" and its 5 families, graphs can share several families.

Scale. MALNET-TINY contains 87,430, Android APK files across 43 types and 246 families of malware. When stored on disk, MALNET-TINY takes over 35 GB of space in edge list format. In Table II, we provide descriptive statistics on the number of nodes, edges, and average degree of MALNET-TINY.

Hierarchy. Android APK contains function call graphs that are assigned a general type (e.g., Fakeapp) and specialized family label (e.g., Artemis) using the Euphony [45] classification structure (see Figure 2). In [45] 4 fields are defined **type** (the kind of threat, i.e., trojan, worm, etc.), **platform** (the OS that the threat is designed to work on, i.e., Windows, Android, etc.), **family** (the group of threats it is associated with in terms of behavior), **information** (extra description of this threat, including its variant). In this paper, we focus specifically on type and family.

Diversity MALNET-TINY offers 43 types, 246 families and graphs averaging 17,588 nodes, 40,105 edges and 2 degrees. The type and family distribution is imbalanced with ratios of 154x and 908x as seen in Figure 1. The graphs have a long-tailed distribution which is difficult to classify because neglecting rare scenarios is likely to result in high-severity errors during our testing. Typically if there are differences in the scales across the input variables it increases the difficulty of the problem being modeled. Figure 1 has large input values the spread of hundreds or thousands of types and families can result in a model that learns large weight values which is an undesired behavior.

Imbalance Models learning from longtailed distributions tend to favor the majority class, leading to poor generaliza-

TABLE II: Graph statistics for MALNET-TINY dataset.

			Nodes				Edges				Avg. Degrees			
Type	#G	#F	min	mean	max	std	min	mean	max	std	min	mean	max	std
Addisplay	17,458	38	37	12,862	97,816	14,556	37	28,072	245,593	33,546	0.92	1.97	4.38	0.37
Spr	13,822	46	12	27,876	168,591	20,943	7	67,389	368,861	51,844	0.58	2.27	4.7	0.44
Spyware	6,590	19	12	5,289	55,409	6,364	7	10,946	121,351	13,954	0.58	1.95	4.27	0.46
Exploit	5,581	13	19	23,842	101,955	14,145	14	45,430	250,498	30,344	0.74	1.88	3.34	0.33
Downloader	4,997	7	37	20,454	106,537	27,811	37	46,397	321,478	63,239	0.96	1.68	3.53	0.66
Smssend++Trojan	4,294	25	16	33,928	146,744	19,122	13	82,441	386,818	47,942	0.81	2.39	3.78	0.23
Troj	3,309	36	14	6,496	64,472	7,698	11	14,846	115,181	18,468	0.79	1.98	5.6	0.52
Smssend	3,266	12	15	19,523	111,452	14,393	12	49,184	337,483	37,847	0.8	2.34	4.61	0.47
Clicker++Trojan	2,867	3	220	6,037	28,970	3,210	471	13,915	71,628	7,321	1.52	2.33	2.92	0.18
Adsware	2,651	16	368	11,462	52,631	12,738	564	25,625	142,795	28,110	1.02	2.19	4.27	0.26
Malware	2,506	19	6	7,543	118,853	12,670	5	16,487	286,126	28,778	0.83	1.9	3.97	0.67
Adware++Adware	2503	2	192	8,845	55,312	6,379	289	19,500	138,170	15,602	1.49	2.16	3.17	0.27
Rog	1,975	22	26	15,154	101,963	19,435	31	34,798	23,1976	46,247	0.91	2.05	4.79	0.49
Spy	1,645	7	48	21,591	106,985	14,830	44	49,261	271,063	39,756	0.92	2.17	3.07	0.25
Monitor	1,357	5	329	3,777	41,236	5,124	580	7,196	102,166	12,044	1.53	1.83	3.09	0.21
Ransom++Trojan	1,153	7	556	5,0924	138,917	22,009	965	114,731	318,871	48,226	1.59	2.26	2.59	0.21
Banker++Trojan	1,106	6	29	33,119	102,744	16,193	36	71,950	236,977	37,509	1.22	2.15	2.99	0.24
Trj	940	18	29	12,675	171,089	16,355	36	29,903	402,259	39,402	1.15	2.2	4.44	0.49
Gray	922	10	51	16,342	65,738	13,446	56	38,880	152,626	31,376	0.88	2.09	4.33	0.58
Adware++Grayware++Virus	835	4	22	5,884	83,833	13,052	20	13,777	193,294	28,681	0.86	2.79	3.17	0.34
Fakeinst++Trojan	718	10	51	14,855	93,646	17,395	58	36,648	229,433	43,608	0.99	2.12	2.84	0.48
Malware++Trj	609	1	52,001	52,107	55,855	596	118,354	118,713	127,640	1,420	2.28	2.28	2.29	0.0
Backdoor	602	10	25	13,485	146,339	21,826	21	33,031	427,245	57,066	0.84	2.19	3.55	0.37
Dropper++Trojan	592	8	47	4,518	67,387	6,734	50	11,355	174,826	18,004	1.06	1.98	3.92	0.7
Trojandownloader	568	7	1,018	38,158	102,061	18,893	1,626	86,200	257,627	44,721	1.34	2.19	2.54	0.21
Hacktool	542	7	668	169,64	40,938	9,263	1,691	37,242	92,124	20,260	1.63	2.21	3.64	0.25
Fakeapp	425	5	24	3,677	49,776	6,830	21	7,886	107,441	15,947	0.88	1.67	2.79	0.37
Clickfraud++Riskware	369	5	1,702	17,730	19,808	2,105	3,647	37,821	42,741	4,621	1.95	2.13	2.25	0.04
Adload	333	4	2,319	18,518	53,009	17,889	4,472	47,829	149,291	47,842	1.46	2.29	3.13	0.4
Addisplay++Adware	294	1	3,253	19,804	49,603	8,739	5,748	41,184	107,862	19,802	1.65	2.03	2.45	0.21
Adware++Virus	274	9	38	14,629	58,735	15,457	38	33,069	138,157	34,872	1.0	2.22	3.17	0.54
Clicker	265	5	47	3,082	75,209	7,333	43	6,369	189,716	17,133	0.91	1.62	3.32	0.51
Fakeapp++Trojan	256	1	44	21,452	72,055	14,505	39	40,628	162,426	33,888	0.88	1.74	2.3	0.27
Riskware++Smssend	247	7	12	2,368	59,834	5,757	7	5,127	154,316	14,257	0.58	1.68	3.0	0.45
Rootnik++Trojan	223	5	210	15,985	83,987	20,863	395	38,766	197,031	50,278	1.15	2.59	3.21	0.47
Worm	220	7	64	14,466	94,028	15,308	78	30,659	203,623	34,250	0.99	1.99	3.42	0.4
Fakeangry	211	2	516	6,000	98,236	11,094	946	14,755	279,102	29,350	1.7	2.35	3.29	0.27
Virus	191	3	681	15,213	79,880	19,075	1,192	34,892	176,862	45,885	1.32	2.12	3.18	0.33
Trojandropper	178	4	220	20,043	78,496	17,769	236	38,829	185,129	38,505	1.03	1.83	4.36	0.32
Adwareare	152	3	893	26,369	57,069	13,751	1,680	60,462	144,487	31,544	1.88	2.25	2.6	0.2
Risktool++Riskware++Virus	152	3	37	16,229	64,558	15,956	37	36,045	158,271	37,407	1.0	1.92	3.17	0.48
Spy++Trojan	119	5	54	31,386	118,239	24,886	66	74,924	293,488	60,837	1.22	2.31	3.26	0.37
Click	113	1	1,821	4,053	11,518	1,984	4,285	7,989	25,508	3,512	1.8	2.04	2.74	0.21

tion performance on rare classes. While class imbalance is traditionally solved by resampling the data (undersampling, oversampling) [4, 46]. Undersampling has been widely used in the class-imbalance learning area. The main deficiency of most existing undersampling methods is that their data sampling strategies are heuristic-based and independent of the used classifier and evaluation metric. Thus, they may discard informative instances for the classifier during the data sampling [46]. The two most common preprocessing techniques are random minority oversampling (ROS) and random majority undersampling (RUS) [47]. In ROS, instances of the minority class are randomly duplicated. In RUS, instances of the majority class are randomly discarded from the dataset. In one of the earliest attempts to improve upon the performance of random resampling, Kubat and Matwin [48] proposed a technique called one-sided selection (OSS). One-sided selection attempts to intelligently undersample the majority class by removing majority class examples that are considered noise, we will not be taking this approach. We will be combining malware minority samples with 'benign' samples in order to predict the evolution of the malware.

IV. Methodology

We propose a methodology to create a malware classifier that provides a degree of robustness to malware polymorphism. Our procedure is composed of four pieces:

- Given a graph representing an android application, extract for each node in the graph the structural vectorial representation via Inferential SIR-GN.
- Use the vectorial representations of all the nodes of a specific graph representing an android application to create the structural pseudo-adjacency matrix. The structural pseudo-adjacency matrix represents the graph, then the android app.
- Combine the structural pseudo-adjacency matrix of the malware with the matrix of the benign android application to create a potential polymorphic version of the malware.
- Use the representations of the android applications (benign and malware) with the representations of the potential polymorphic version of the malware to train a random forest algorithm to identify and classify malware.

In the following, we describe the Inferential SIR-GN, the structural pseudo adjacency matrix, and the matrix combina-

tion for potential polymorphic combination.

A. Inferential SIR-GN

The algorithm Inferential SIR-GN, used for extracting node representations from the directed graph, is described in detail in Layne and Serra [49]. The model relies upon the methodology of SIR-GN, first described in [50], wherein a node's representation is iteratively updated by describing then aggregating its neighbors. The size of a node's representation at each iteration is equal to a user-chosen hyperparameter nc. Node descriptions are generated by clustering the current node description (which initializes as the node degree) into nc KMeans clusters. Normalization of the representation occurs before the clustering step at each iteration, then the distance from each cluster centroid is converted into a probability of membership of the node in each cluster. Once a node's structural description has been updated, its neighbors are aggregated into its description by summing for each cluster all neighbors' probabilities of membership per cluster. The resulting node representation is equal to the expected number of neighbors that node possesses in each cluster. Each iteration corresponds to an added depth of exploration, where k iterations will generate a node description incorporating the k-hop neighborhood structure of a node.

Inferential SIR-GN differs from the standard model via multiple modifications, the first being that at the end of each iteration, we concatenate each node's structural description into a larger representation that captures the evolution of the structural information through deeper neighborhood exploration. After the final iteration, a Principle Component Analysis (PCA) is used to prevent degradation of the information as the representation size grows. The final representation is condensed to a size chosen as a hyperparameter. For directed graphs, a node's initial representation begins as two vectors of size nc, one containing the node's in-degree, the other containing its out-degree. These two are concatenated together before clustering. At each iteration, clustering of this larger node vector is performed, followed by aggregation of the neighbors. For directed data, the aggregation is performed separately for a node's in-neighbors and out-neighbors into two intermediate vectors, then once again concatenated together for the next iteration. Inferential capability of our proposed model is accomplished by pre-training the KMeans and scalers for each iteration - a new KMeans and Scaler are used for every depth of exploration - along with the PCA model that will be used to generate the final node representation. We pre-train on random graphs and store each model for use in inference. At inference time, repeated normalization followed by clustering and aggregation is accomplished using the pre-trained models, and the PCA fit during training is used to generate the final node representations. This drastically increases inference time, and the same pre-trained model can be used on a variety of different data sources. This is demonstrated extensively in Layne and Serra, along with a detailed algorithm and description of the time complexity of the model.

B. Structural Pseudo-Adjacency Matrix

Given the vectorial representation of SIR-GN, [49] provides a procedure to create a unique graph representation technique. Such techniques identify groups of nodes in a fixed number. Each group contains nodes with similar vectorial representations. Given this set of groups, the method creates a structural pseudo-adjacency matrix working on the groups that, once flattened, represents the vectorial representation of the graph. Then, the vectorial representations of the two graphs are comparable if the computation of the node representations and the definition of the groups of nodes for the structural pseudo-adjacency matrices are identical for the two graphs. This approach guarantees this property because inferential SIR-GN [49] is a procedure able to perform inferences and that is pretrained on a specific family of directed random graphs. Note that since the groups are created on the basis of structural similarities among the nodes, the graph representation is invariant.

More specifically, these node representations are used to train a final scalar and KMeans, that clusters the entire graph data at inference time. Unlike the incremental KMeans, which only see the node representation/aggregation for the current level of depth being explored, this final KMeans is fit using the concatenated iterative node representations condensed by PCA. During inference, the nodes of the target graph are embedded as described above, then clustered one final time using the KMeans pre-trained on the full graph data. The distances to the cluster centroids are transformed into probabilities of membership in a cluster, as above. However, the aggregation method is markedly different for graph representation than for nodes. Graph representations are often created from node representations by sum or mean-pooling node representations. In Layne and Serra [49], a new method is presented for node pooling, which creates a structural pseudo-adjacency matrix of the size $ngc \times ngc$, where the matrix is the sum of each node vector multiplied by the transpose of each of its neighbors. This creates a matrix that is not only unique to a specific graph structure but also indifferent to node ordering, unlike typical adjacency matrices. The linearized matrix yields a set of features for use in downstream graph classification tasks.

C. Malware Polymorphic Generation Approach

By using the structural pseudo-adjacency matrix representing each android application, we design a procedure to create a polymorphic version of existent malware. Given an android application a, $SPAM(a)$ denotes the structural pseudo-adjacency matrix of the graph of the application a. The procedure for each android malware m, searches among the benign applications the application b which has $SPAM(b)$ close in terms of euclidean distance to $SPAM(a)$. The computation of the closest benign application is performed fast by using the k-nearest neighbor algorithm. Given the android malware a and the closets benign application b, a polymorphic

representation pr_a of a is obtained by using the following weighted mean of the two representations:

$$pr_a = 0.8 \cdot SPAM(a) + 0.2 \cdot SPAM(b)$$

The weights are used in order to slightly modify the malware representation $SPAM(a)$. Then for each malware, its polymorphic representation is created, then all the polymorphic representations are used in the training of the classification model, in our case a random forest, to make the classification model robust to polymorphic changes of the malware applications.

V. Experiments

A. Experimental Setup

For type level classification experiments, MALNET-TINY 5k is split into 3,500 training, 500 validation, and 1,000 graphs. MALNET-TINY is split into 61,201 training, 8,743 validation and 17,486 graphs. MALNET-TINY and MALNET-TINY 5k are split for type-level classification experiments and evolution prediction classification. We compare against Freitas, Duggal, and Chau [1], using a ResNet18 model, trained from scratch on grayscale images using cross-entropy loss and class reweighting where the model achieves a macro-F1 score of 0.651, macro-precision of 0.672, and a macro-recall of 0.646 [1]. We will also analyze MALNET-TINY and MALNET-TINY 5k by performing type-level classification experiments on different data splits (1) A random split into 61,201 training, 8,743 validation, and 17,486 graphs. (2) A temporal split where training data contains 69,944 and 17,86 test graphs. (3) A temporal split and we apply obfuscation methods to increase our model's performance. Each dataset is transformed with the SIR-GN method which encodes node structure and applies an iterative process that takes advantage of node clustering and node neighborhood relations in order to learn rich structural representations [5]. SIR-GN's final output is a structural representation vector fed into an XGBoost classifier. Every model is evaluated on its macro-F1 score, we also provide additional performance metrics such as precision, and recall.

B. Empirical Results

We present results for SIR-GN graph representation technique [49] found in III and results for MALNET-TINY 5k found https://mal-net.org/ and MALNET-TINY split into 61,201 training, 8,743 validation and 17,486 test graphs [1]. We perform our experiments in Python3 using an Intel (R) Core(TM)i7-7700HQ CPU @ 2.80 GHz.

We use the following two datasets:

- **MALNET-TINY 5k** dataset contains 5 types Addisplay, Adware, Benign, Downloader, and Trojan with each containing 1000 graphs.
- **MALNET-TINY 61K** 61,201 training, 8,743 validation and 17,486 test graphs for type level classification experiments.
- **MALNET-TINY 81K** 81,201 training, 8,743 validation and 27,486 test graphs for type level classification experiments and evolution prediction.

TABLE III: We compare SIR-GN driven model against ResNet18 using MALNET-TINY 60k. The ResNet18 model is outperformed.

Model	Type		
	F1	Precision	Recall
ResNet18	0.651	0.672	0.646
SIR-GN XGB	**0.718**	**0.729**	**0.794**

TABLE IV: Evaluating the performance of three different dataset splits containing 5,000 graphs (MALNET-TINY 5k). SIR-GN algorithm produces the following macro-F1, macro-precision, and macro-recall. Performance is similar across data splits.

Tiny Dataset	Binary			Type		
	F1	Precision	Recall	F1	Precision	Recall
Random TINY-5k	0.851	0.832	0.876	0.916	0.917	0.915
Temporal TINY-5k	0.716	0.716	0.777	0.725	0.739	0.807
Evolution TINY-5k	0.752	0.744	0.807	0.741	0.740	0.819

TABLE V: Comparison of Inferential SIR-GN driven model in MALNET-TINY 5k against several other graph based methods reported in [51]

Model	Type Accuracy
SIR-GN XGB	**0.92**
Feather [52]	0.86
LDP [53]	0.86
GIN [54]	0.90
GCN [55]	0.81
Slaq-LSD [56]	0.76
NoG [57]	0.77
Slaq-VNGE [56]	0.53

TABLE VI: Evaluating the performance of three different dataset splits containing 87,430 graphs (MALNET-TINY 81k). SIR-GN algorithm produces the following macro-F1, macro-precision, and macro-recall. Performance is similar across data splits.

Tiny Dataset	Binary			Type		
	F1	Precision	Recall	F1	Precision	Recall
Random TINY 81K	0.468	0.605	0.471	0.741	0.813	0.702
Temporal TINY 81K	0.474	0.648	0.466	0.725	0.739	0.807
Evolution TINY 81K	0.476	0.653	0.496	0.741	0.740	0.819

Table III contains the results for the MALNET-TINY dataset containing macro-F1, macro-precision, and macro-recall on random splits of 61,201 training graphs, 8,743 validation graphs, and 17,486 test graphs. We compared against [1] where they achieved a macro-F1 score of 0.651, macro-precision of 0.672,and a macro-recall of 0.646 and we achieve a macro-F1

score of 0.718, macro-precision of 0.729,and a macro-recall of 0.794. Using the SIR-GN algorithm improves the performance of the classifier as seen by the results.

Table IV contains the results for the MALNET-TINY 5k dataset containing macro-F1, macro-precision, and macro-recall on random splits of 3,500 training graphs, 500 validation graphs, and 1000 test graphs. Table IV contains the macro-F1 score, macro-precision, and a macro-recall for random splits. For the random split we get a macro-F1 score of 0.916, a macro-precision of 0.917, and a macro-recall of 0.915. [51] provides accuracy scores for several graph based methods, as seen in Table V, the Inferential SIR-GN approach stands out as the best at 0.92 accuracy. The table also has scores for a temporal split where the training consisted of a date range from 2012 to 2019 and the test date range was from 2020 to 2021. For the temporal split and type classification, we achieved a macro-F1 score of 0.725, a macro-precision of 0.739, and a macro-recall of 0.807. When we apply evolution prediction we get a performance boost for a macro-F1 score of 0.741 vs 0.725.

In Table VI we evaluate MALNET-TINY and include benign malware type to generate a score on a random split. This table contains the macro-F1 score, macro-precision, and a macro-recall for random a split and temporal split. The table also has scores for a temporal split where the training consisted of a date range from 2012 to 2019 and the test date range was from 2020 to 2021. For the temporal split and type classification, we achieved a macro-F1 score of 0.725, a macro-precision of 0.739, and a macro-recall of 0.807. When we apply evolution prediction we get a performance boost for a macro-F1 score of 0.741 vs 0.725.

VI. Conclusion

In this work we focused on MALNET and highlighted graph representation learning methods which have emerged across many scientific fields and are driving the development of representation learning techniques. Graph representation learning techniques encode structured information into low dimensional space for a variety of important downstream tasks. We proposed a methodology to create a malware classifier that provides a degree of robustness to malware polymorphism. Given a graph representing an android application, we extracted for each node in the graph the structural vectorial representation via Inferential SIR-GN. In this research we used the representations of the android applications (benign and malware) with the representations of the potential polymorphic version of the malware to train a model to identify and classify malware. Experimental results

In the experimental setup and we cover the empirical results. We present results for SIR-GN graph representation technique. We find that Malware applications are dangerous and create huge damages. Therefore, the detection and classification of malware are essential for their mitigation. Transforming binary executables into images represents a feasible and accurate way to detect and classify malware with image neural networks. Differently from the Image-based approach, in this work, we proposed a procedure based on control flow graph, structural graph representation learning, and XGboost. Such a procedure was tested on MALNET in terms of malware classification and provided better results than the one image-based working using the neural network ResNet. In addition, by using the structural pseudo-adjacency matrix representing each android application, we designed a procedure to create a polymorphic version of existent malware application to improve the training phase and the classification performances.

Acknowledgment

This research was made possible by the National Science Foundation award #1820685 and Idaho Global Entrepreneurial Mission/Higher Education Research Council #IGEM22-001.

References

[1] S. Freitas, R. Duggal, and D. H. Chau, "Malnet: A large-scale cybersecurity image database of malicious software," *CoRR*, vol. abs/2102.01072, 2021. [Online]. Available: https://arxiv.org/abs/2102.01072

[2] S. Freitas, A. Wicker, D. H. Chau, and J. Neil, "D2M: dynamic defense and modeling of adversarial movement in networks," *CoRR*, vol. abs/2001.11108, 2020. [Online]. Available: https://arxiv.org/abs/2001.11108

[3] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, ser. VizSec '11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/2016904.2016908

[4] S. Freitas, Y. Dong, J. Neil, and D. H. Chau, "A large-scale database for graph representation learning," 2020.

[5] M. Joaristi and E. Serra, "Sir-gn: A fast structural iterative representation learning approach for graph nodes," *ACM Trans. Knowl. Discov. Data*, vol. 15, no. 6, May 2021. [Online]. Available: https://doi.org/10.1145/3450315

[6] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," 2018.

[7] A. S. Bozkir, A. O. Cankaya, and M. Aydos, "Utilization and comparision of convolutional neural networks in malware recognition," in *2019 27th Signal Processing and Communications Applications Conference (SIU)*, 2019, pp. 1–4.

[8] T. M. Mohammed, L. Nataraj, S. Chikkagoudar, S. Chandrasekaran, and B. S. Manjunath, "Malware detection using frequency domain-based image visualization and deep learning," *CoRR*, vol. abs/2101.10578, 2021. [Online]. Available: https://arxiv.org/abs/2101.10578

[9] L. Chen, R. Sahita, J. Parikh, and M. Marino, "Stamina: Scalable deep learning approach for malware classification," 2020.

[10] J. Gennissen and J. Blasco, "Gamut : Sifting through images to detect android malware," 2017.

[11] K. Kancherla and S. Mukkamala, "Image visualization based malware detection," *2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, pp. 40–44, 2013.

[12] S. Choi, S. Jang, Y. Kim, and J. Kim, "Malware detection using malware image and deep learning," in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, 2017, pp. 1193–1195.

[13] H. Zhang, X. Xiao, F. Mercaldo, S. Ni, F. Martinelli, and A. Kumar, "Classification of ransomware families with machine learning based on n -gram of opcodes," *Future Generation Computer Systems*, vol. 90, 08 2018.

[14] J. Su, D. V. Vasconcellos, S. Prasad, D. Sgandurra, Y. Feng, and K. Sakurai, "Lightweight classification of iot malware based on image recognition," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 02, 2018, pp. 664–669.

[15] I. Yoo, "Abstract visualizing windows executable viruses using self-organizing maps."

[16] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114–123, 2016.

[17] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck, "Structural detection of android malware using embedded call graphs," in *In Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security (AISec'13*, 2013.

[18] S. Ranveer and S. Hiray, "Comparative analysis of feature extraction methods of malware detection."

[19] M. Nunes, "Dynamic Malware Analysis kernel and user-level calls," Mar. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1203289

[20] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," 2021.

[21] X. Yan, H. Cheng, J. Han, and P. Yu, "Mining significant graph patterns by leap search," in *SIGMOD 2008*, ser. Proceedings of the ACM SIGMOD International Conference on Management of Data, Dec. 2008, pp. 433–444, 2008 ACM SIGMOD International Conference on Management of Data 2008, SIGMOD'08 ; Conference date: 09-06-2008 Through 12-06-2008.

[22] X. Kong and P. Yu, "Multi-label feature selection for graph classification," 12 2010, pp. 274–283.

[23] H. NT, C. J. Jin, and T. Murata, "Learning graph neural networks with noisy labels," 2019.

[24] A. Lusci, G. Pollastri, and P. Baldi, "Deep architectures and deep learning in chemoinformatics: The prediction of aqueous solubility for drug-like molecules," *Journal of chemical information and modeling*, vol. 53 7, pp. 1563–75, 2013.

[25] K. Riesen and H. Bunke, "Iam graph database repository for graph based pattern recognition and machine learning," in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 2008, pp. 287–297.

[26] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.

[27] X. Yue, Z. Wang, J. Huang, S. Parthasarathy, S. Moosavinasab, Y. Huang, S. M. Lin, W. Zhang, P. Zhang, and H. Sun, "Graph embedding on biomedical networks: methods, applications and evaluations," *Bioinformatics*, vol. 36, no. 4, pp. 1241–1251, 10 2019. [Online]. Available: https://doi.org/10.1093/bioinformatics/btz718

[28] B. Rozemberczki, O. Kiss, and R. Sarkar, "Karate club: An api oriented open-source python framework for unsupervised learning on graphs," 2020.

[29] ——, "Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs," in *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*. ACM, 2020, p. 3125–3132.

[30] L. Nataraj, P. Porras, and V. Yegneswaran, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis abstract."

[31] M. Ceci, A. Cuzzocrea, and D. Malerba, "Supporting roll-up and drill-down operations over olap data cubes with continuous dimensions via density-based hierarchical clustering." in *SEBD*. Citeseer, 2011, pp. 57–65.

[32] E. Serra, M. Joaristi, and A. Cuzzocrea, "Large-scale sparse structural node representation," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 5247–5253.

[33] P. Braun, A. Cuzzocrea, T. D. Keding, C. K. Leung, A. G. Padzor, and D. Sayson, "Game data mining: clustering and visualization of online game data in cyber-physical worlds," *Procedia Computer Science*, vol. 112, pp. 2259–2268, 2017.

[34] A. Guzzo, D. Sacca, and E. Serra, "An effective approach to inverse frequent set mining," in *2009 Ninth IEEE International Conference on Data Mining*. IEEE, 2009, pp. 806–811.

[35] K. J. Morris, S. D. Egan, J. L. Linsangan, C. K. Leung, A. Cuzzocrea, and C. S. 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 *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018, pp. 1486–1491.

[36] E. Serra and 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*, vol. 1, no. 1, pp. 66–88, 2014.

[37] L. Bellatreche, A. Cuzzocrea, and 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. Springer, 2010, pp. 89–104.

[38] O. Korzh, M. Joaristi, and E. Serra, "Convolutional neural network ensemble fine-tuning for extended transfer learning," in *International Conference on Big Data*. Springer, 2018, pp. 110–123.

[39] S. Ahn, S. V. Couture, A. Cuzzocrea, K. Dam, G. M. Grasso, C. K. Leung, K. L. McCormick, and 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 (FUZZ-IEEE)*. IEEE, 2019, pp. 1–6.

[40] E. Serra, A. Sharma, M. Joaristi, and O. Korzh, "Unknown landscape identification with cnn transfer learning," in *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2018, pp. 813–820.

[41] E. Serra, A. Shrestha, F. Spezzano, and A. Squicciarini, "Deeptrust: An automatic framework to detect trustworthy users in opinion-based systems," in *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, 2020, pp. 29–38.

[42] M. Joaristi, E. Serra, and F. Spezzano, "Inferring bad entities through the panama papers network," in *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2018, pp. 767–773.

[43] ——, "Detecting suspicious entities in offshore leaks networks," *Social Network Analysis and Mining*, vol. 9, no. 1, pp. 1–15, 2019.

[44] G. Sood, *virustotal: R Client for the virustotal API*, 2017, r package version 0.2.1.

[45] M. Hurier, G. Suarez-Tangil, S. K. Dash, T. F. Bissyandé, Y. Le Traon, J. Klein, and L. Cavallaro, "Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 425–435.

[46] M. Peng, Q. Zhang, X. Xing, T. Gui, X. Huang, Y.-G. Jiang, K. Ding, and Z. Chen, "Trainable undersampling for class-imbalance learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 4707–4714, Jul. 2019. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/4396

[47] J. Van Hulse, T. Khoshgoftaar, and A. Napolitano, "Experimental perspectives on learning from imbalanced data," vol. 227, 01 2007, pp. 935–942.

[48] M. Kubat, "Addressing the curse of imbalanced training sets: One-sided selection," *Fourteenth International Conference on Machine Learning*, 06 2000.

[49] J. Layne and E. Serra, "Inferential sir-gn: Scalable graph representation learning," 2021.

[50] M. Joaristi and E. Serra, "Sir-gn: A fast structural iterative representation learning approach for graph nodes," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 15, no. 6, pp. 1–39, 2021.

[51] S. Freitas, Y. Dong, J. Neil, and D. H. Chau, "A large-scale database for graph representation learning," 2021.

[52] B. Rozemberczki and R. Sarkar, "Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models," 2020.

[53] C. Cai and Y. Wang, "A simple yet effective baseline for non-attributed graph classification," 2019.

[54] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" 2019.

[55] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[56] A. Tsitsulin, M. Munkhoeva, and B. Perozzi, "Just slaq when you approximate: Accurate spectral distances for web-scale graphs," *Proceedings of The Web Conference 2020*, Apr 2020. [Online]. Available: http://dx.doi.org/10.1145/3366423.3380026

[57] T. H. Schulz and P. Welke, "On the necessity of graph kernel baselines," 2019.