Representation Learning with Function Call Graph Transformations for Malware Open Set Recognition

Jingyun Jia
Department of Computer Engineering and Sciences
Florida Institute of Technology
Melbourne, FL 32901, US
Email: jiaj2018@my.fit.edu

Philip K. Chan
Department of Computer Engineering and Sciences
Florida Institute of Technology
Melbourne, FL 32901, US
Email: pkc@cs.fit.edu

***Abstract*—Open set recognition (OSR) problem has been a challenge in many machine learning (ML) applications, such as security. As new/unknown malware families occur regularly, it is difficult to exhaust samples that cover all the classes for the training process in ML systems. An advanced malware classification system should classify the known classes correctly while sensitive to the unknown class. In this paper, we introduce a self-supervised pre-training approach for the OSR problem in malware classification. We propose two transformations for the function call graph (FCG) based malware representations to facilitate the pretext task. Also, we present a statistical thresholding approach to find the optimal threshold for the unknown class. Moreover, the experiment results indicate that our proposed pre-training process can improve different performances of different downstream loss functions for the OSR problem.**

***Index Terms*—Malware classification, open set recognition, self-supervised learning, representation learning**

I. Introduction

As machine learning has achieved great success in various domains, there is still a wide range of challenges in the real world. For example, from the security scenario, new malware classes emerge daily. A robust machine learning system for malware detection should be able to classify the known malware classes and recognize the newly unknown malware classes, which is referred as Open Set Recognition (OSR) problem [1]. The OSR problem aims to classify the multiple known classes for a multinomial classification problem while identifying the unknown classes.

In this paper, we follow a two-stage learning approach to address the OSR problem in malware classification. Given the malware assembly files, we first extract the function call graphs (FCGs) as the input representations of the malware samples. Next, to learn better representations for the malware samples, we use a self-supervised pre-training approach for the extracted FCGs. As the self-supervised learning approach needs a pretext task, we propose two transformations for the FCG inputs. Then both original and transformed FCGs are processed by a detransformation autoencoder (DTAE) [2]. DTAE involves an encoder and a decoder. The encoder learns the representations for the inputs while the decoder reconstructs the transformed inputs back to their original forms. After pre-training and fine-tuning the representations, we apply a statistical thresholding approach to find the optimal threshold for the OSR tasks. Our contributions include, first, we summarize the characteristics of the malware FCGs. Second, we propose two transformation methods for the malware FCGs to facilitate the self-supervised pre-training process for the OSR tasks. Third, we introduce a statistical thresholding approach for the OSR task, which performs similarly to the manually selected threshold. Finally, our experiments on two different malware datasets indicate that our proposed self-supervised pre-training approach improves the model performance on the OSR tasks.

We organize this paper as follows. In section 2, we review some related research works. In section 3, we first present our proposed approach to the self-supervised pre-training for the malware FCGs, then introduce a statistical thresholding approach for the OSR tasks. Finally, section 4 evaluates the proposed approach through experiment setup and results from the analysis.

II. Related Work

Function Call Graphs The graph features can preserve the structural information between different entities, and have been widely used in many research fields, such as social network recommendation [3], molecules structure study [4] and malware classification [5]. Specifically, the researches in [6] and [5] extract function call graphs (FCGs) from disassembled binary files. An FCG is a directed graph where the vertices represent the function clusters (procedures), and the edges represent the caller-callee relation between the functions (vertices). As the FCGs have a good performance in saving the interaction information between functions, in this work, we also use malware FCGs as input features for the open set recognition (OSR) problem.

Open Set Recognition The objective of the OSR problem is to classify the multiple known classes for a multinomial classification problem while identifying the unknown classes. As new and unknown malware class occurs regularly, it is impossible to collect samples that exhaust all the malware classes. An advanced malware classification system should adapt to the open set scenario, classifying the known classes while recognizing the unknown class. Recent work have brought neural network-based approach for the OSR problem such as the works in [1], [7] and [8]. OpenMax [1] adapts Extreme Value Theory (EVT) meta-recognition calibration in

the penultimate layer of the networks. Further, it redistributes values of the activation vector to estimate the probability of an input being from an unknown class. Hassen and Chan propose ii loss for open set recognition [7]. It first finds the representations for the known classes during training and then recognizes an instance as unknown if it does not belong to any known classes. MMF [8] is an extension to different types of loss functions (classification loss and representation loss) to facilitate the OSR task. It further separates the known and unknown representations by increasing the signature feature magnitudes of the known classes. Here, we propose a self-supervised learning approach for the malware OSR problem. Adding such a self-supervised pre-training process makes classification loss and representation loss functions more sensitive to the unknown class.

Self-supervised Learning Self-supervised learning uses a pretext task that is different from the primary task to learn the representations. The pretext task includes autoencoding, classifying transformations such as rotations [9], intra-sample vs inter-sample transformations in contrastive loss [10], redundancy reduction in learned features from transformations [11]. In addition to image recognition applications, more recent research has extended self-supervised learning to graph representation learning. Specifically, Graph contrastive learning (GraphCL) in [12] designs four types of transformations for a graph contrastive learning framework: node dropping, edge perturbation, attribute masking, and subgraph sampling. The experimental results indicate that the beneficial graph transformation technique is dataset-specific. Moreover, Pairwise Half-graph Discrimination (PHD) in [13] proposes self-supervised multi-scale contrastive learning for graph representation learning. The approach first generates two augmented views based on local and global perspectives from the input graph. Then, the objective function maximizes the agreement between node representations across different views and networks. However, as we will discuss later in Section III-B, FCGs are sparser than most graph datasets, such as social networks. The existing graph transformation techniques like node dropping and subgraph sampling are less applicable to FCGs. Our work here introduces two different transformations for the malware FCGs inputs. And then, we adopt the same learning strategy as in DTAE [2], i.e., reconstructing the transformed inputs back to original forms to improve the quality of learned representations.

III. Approach

The objective of open set recognition (OSR) is to classify the known classes and the unknown classes even when the collected training samples cannot exhaust all the classes. An advanced malware classification system that utilizes OSR techniques can classify the known malware families while identifying the unknown malware family. Hassen and Chan [5] convert malware assembly files to FCGs as OSR input. Here, we also use the FCGs as input samples. To learn better representations for the OSR problem in malware classification, we introduce a self-supervised pre-training process to learn low-level features of the malware samples. Based on the FCGs characteristics, we propose two transformation methods for malware FCGs to facilitate the pretext task. Moreover, we introduce a statistical method to identify unknown instances.

Table I: Graph statistics for datasets in function call graphs (FCGs), biochemical molecules (BMs) and social networks (SN). The statistics includes: average number of vertices, average number of degrees and % of vertices that are neighbors (Degree/Vertex), average number of connected components (C.C.), average size of each connected components and relative connected components size (C.C. Size/Vertex).

Dataset	Category	Vertex	Degree (/Vertex)	C.C.	C.C. Size (/Vertex)
MS	FCGs	27.55	1.66(6%)	14.99	3.74(16%)
AG	FCGs	31.73	3.31(10%)	16.97	2.28(7%)
MUTAG	BMs	17.93	1.10 (6%)	3.49	5.86(33%)
PROTEINS	BMs	39.06	1.86(5%)	4.75	9.78(25%)
COLLAB	SNs	74.49	32.99(44%)	4.65	30.36(41%)
DBLP_v1	SNs	10.48	1.87(18%)	1.93	6.12(58%)

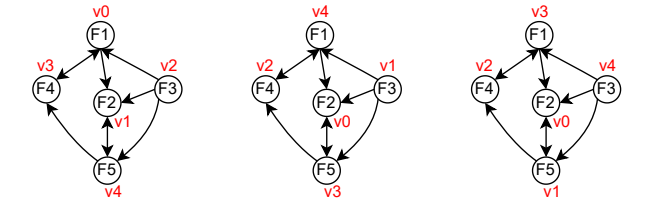
A. Malware Function Call Graphs (FCGs)

Previous research works have proposed various ways to extract features for malware classifications: Schultz et al. [14] extract features from printable strings in malware binaries. Hu et al. [15] extract features from instruction n-grams. Hassen and Chan [5] convert malware assembly files to FCGs as input features. The FCGs can better preserve structural information between functions. Thus, in this paper, we adopt the same FCGs as in [5]. The system first extracts FCG representations from dissembled binaries. In the FCGs, the vertices are functions, and edges are the interactions (calls) between functions. Then based on the instruction opcode sequence, it clusters the functions using Locality Sensitive Hashing (LSH), and the vertices (functions) are then arbitrarily labeled with cluster-ids.

The extracted FCGs are directed graph representations of the dissembled malware binaries, with function clusters as the graph vertices and the caller-callee relations between functions as graph edges. As the cluster ids are arbitrarily assigned, we will get different isomorphic graphs for the same malware binaries when we change the order of the cluster ids.

B. FCG characteristics

In this subsection, we compare the characteristics of the FCGs of malware datasets with two other categories of graphs: biomedical molecules (BMs) and social networks (SNs) in Table I. Specifically, we compare the FCGs extracted from two malware datasets: Microsoft Challenge (MC) and Android Genome (AG) (see section IV for more details) with MUTAG [16], PROTEINS [17], COLLAB [18] and DBLP_v1 [19]. In the table, "Vertex" and "Degree" are the average numbers of vertices and degrees in each dataset. We also measure the average percentage of vertices that are neighbors by dividing the number of degrees by the number of vertices. Moreover, we calculate the average number of connected components (C.C.) and the average size of connected components (C.C. Size) for each dataset. Also, we divide the size of the C.C. by the number of vertices to measure the relative C.C. Size.



(a1) (b1) (c1)

	F1	F2	F3	F4	F5
F1	0	1	0	1	0
F2	0	0	0	0	1
F3	1	1	0	0	1
F4	1	0	0	0	0
F5	0	1	0	1	0

	F2	F3	F4	F5	F1
F2	0	0	0	1	0
F3	1	0	0	1	1
F4	0	0	0	0	1
F5	1	0	1	0	0
F1	1	0	1	0	0

	F2	F5	F4	F1	F3
F2	0	1	0	0	0
F5	1	0	1	0	0
F4	1	0	0	1	0
F1	1	0	1	0	0
F3	1	1	0	1	0

(a2) (b2) (c2)

(a) Original (b) FCG-shift (c) FCG-random

Figure 1: Transformations of FCG adjacency matrix

Comparing the graph statistics of the FCGs with the other categories, we conclude two characteristics of the FCGs.

First, FCGs are sparser (i.e., have fewer direct neighbors) than the graphs from the other two categories, especially social networks. In the COLLAB dataset, the average degree of a graph is 32.99, which means 44% of the vertices are direct neighbors. Meanwhile, 6% of vertices are direct neighbors in the MS dataset and 10% for the AG dataset.

Second, FCGs have more and relatively small connected components than the other two categories of graphs. From Table I, both malware FCGs contain around 15 connective components, while the datasets from the other two categories contain less than five connected components. Furthermore, the average sizes of each connected component in the two malware FCGs are less than 4, which means less than four vertices are connected while isolated from the rest of the vertices. Especially for the AG dataset, the connected components are of size 2.28 on average, which is only 7% of the total vertices. The relative connected components size is above 25% of the total vertices for the four datasets from the other two categories. Notably, the relative size of connected components in DBLP_v1 dataset reaches 58%.

C. FCG transformations

Self-supervised learning generally involves input transformations to achieve pretext tasks to learn better representations of input samples. The research in [12] finds that the optimal input transformation method is task-relevant, and it concludes that node dropping and subgraph sampling are generally beneficial across biochemical molecules and social networks datasets. The node dropping transformation creates a new graph view by discarding a specific set of vertices and edges from the original input graph. As the FCGs have fewer direct neighbors and are sparser than other graph datasets, discarding vertices and their edges will remove more neighborhood information. Thus the node dropping transformation is less applicable to the malware FCGs. The subgraph sampling transformation creates a new graph view by sampling a subgraph from the original input graph via a random walk. From the second characteristic of the FCGs, the FCGs contain more connected components (around 15 for the FCGs dataset from Table I). Since a random walk subgraph sampling will keep one connected component and discard the rest (14 out of 15), the subgraph sampling will discard more than 90% information. Thus subgraph sampling is not desirable in learning the representations of the FCGs.

As FCGs can be represented by adjacency matrices, and the ordering of vertices in the matrices is arbitrary. Here, we propose two types of transformations: FCG-shift and FCG-random for the malware FCGs. The two transformations generate a new isomorphic view by altering the ordering of vertices. Given the original order of clusters-ids assignment as Figure 1a, the FCG-shift transformation randomly select a pivots n, and then shift the cluster-ids assignments n positions to the left. For example, in Figure 1b, the order of vertices (cluster-ids) is shifted one position to the left. The original vertex order "F1", "F2", "F3", "F4", "F5" becomes "F2", "F3", "F4", "F5", "F1". The FCG-random transformation randomly permute the order of vertices and generated new adjacency matrices based on the permuted vertex order. In Figure 1c, after the random permutation, the original vertex order "F1", "F2", "F3", "F4", "F5" becomes "F2", "F5", "F4", "F1", "F3". Both FCG-shift and FCG-random maintain the orignal FCGs' properties without information loss by generating isomorphic graphs to the original graphs.

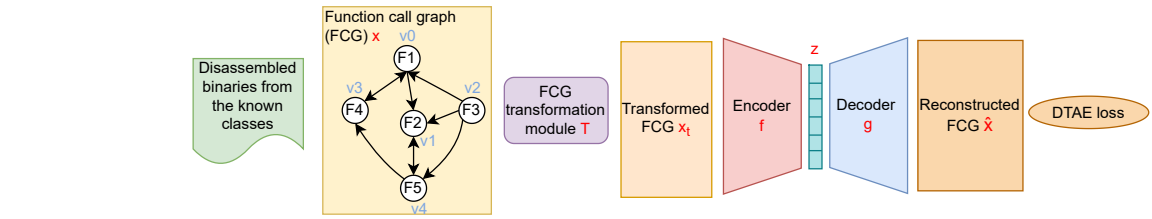
D. Representation Learning

In this work, we follow the two-stage learning strategy to learn the representations of input malware FCGs. We adopt the self-supervised learning strategy to initial the network with low-level representations in the first stage. In the second stage, we fine-tune the network with different loss functions to extract the discriminative representations.

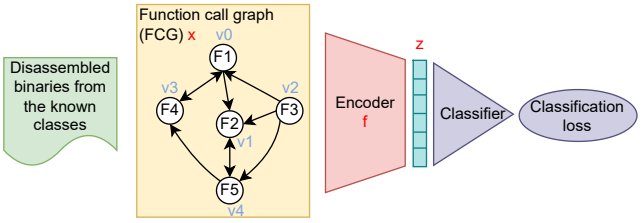
1) Pre-training stage: With the proposed FCG transformations, we adopt detransformation autoencoder (DTAE) proposed in [2] as our pretext task here to pre-train the the network. As depict in Figure 2a, given an input disassembled binaries of the malware samples from the known classes, we first extract its FCG x_i. Then the FCG transformation module $T(.)$ transforms the original FCG to its correlated views x_{it}. Next, the encoder $f(.)$ learns the representations z of the transformed FCG x_{it}, and the decoder reconstruct the representation z back to its original FCG format $\hat{x}_{it}$. Assuming we have M transformations for N FCG inputs. The learning process of neural network-based encoder-decoder structure is guided by DTAE loss:

$$\mathcal{L}_{\text{DTAE}} = \frac{1}{2}\sum_{t=0}^{M-1}\sum_{i=1}^{N}(x_i - \hat{x}_{it})^2 \tag{1}$$

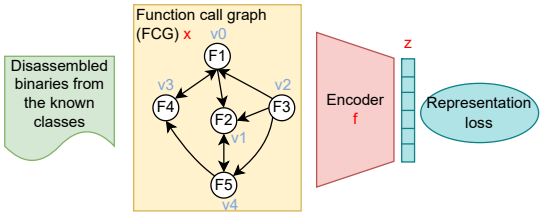
In this paper, we transform FCGs four times for each experiment, i.e., $M = 4, t \in \{0, 1, 2, 3\}$.



(a) Pre-training step



(b) Fine-tuning step with classification loss



(c) Fine-tuning step with representation loss

Figure 2: The training process of using detransformation autoencoder.

2) Fine-tuning stage: After pre-training the neural network with transformed inputs, we fine-tune the encoder and presentation layer (z) with the original inputs for the downstream tasks. Here, we consider two types of loss functions for fine-tuning: classification loss and representation loss. The objective of classification loss is to explicitly lower the training data's classification error in the decision layers, such as cross-entropy loss. When using classification loss as the fine-tuning loss function, we connect the presentation layer with a classifier, which associates with a classification loss function as shown in Figure 2b. The objective of representation loss functions is to learn better representations of training data. The representation loss functions are normally applied to the representation layers, such as triplet loss. When using representation loss as the fine-tuning loss function, we directly constrain the representation layer with a representation loss function, as shown in Figure 2c.

E. Open Set Recognition

After fine-tuning the encoder with the original FCG inputs, we extract the learned representations z for the malware input. We utilize the distances between the representations for the open set recognition (OSR) task: classifying the known classes and identifying the unknown class.

For a known class k that participant in the training process, we first find its representation centroid as prototype μ_k. Given the representation z_i for sample i from class k (i.e. $y_i = k$), we can calculate the prototype as:

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N_k} z_i, \tag{2}$$

where N_k is the number of samples in class k. After obtaining the prototypes, we introduce a statistical method to perform the OSR task. Specifically, we calculate the mean m_k and standard deviation s_k of the distances d_i from the training samples to the prototype k.

$$m_k = \frac{1}{N_k} \sum_{i=1}^{N_k} d_i \tag{3}$$

$$s_k = \sqrt{\frac{\sum_{i=1}^{N_k} (z_i - m_k)^2}{N_k}} \tag{4}$$

Then we normalize the distances between representations and prototypes based on the prototypes' means and standard deviations, and calculate the outlier score based on the least of standard deviations to the prototype :

$$outlier\_score(x) = \min_{1 \le k \le C} \frac{\|D(\mu_k, z) - m_k\|}{s_k}, \tag{5}$$

where C is the number of known classes, and z is the learned representation of input x. $D(.,.)$ is a distance function, and we use euclidean distances in this paper. Based on the Empirical Rule, a test instance can be recognized as "unknown" if its outlier score is more significant than three standard deviations.

$$y = \begin{cases} unknown, & \text{if } outlier\_score(x) > 3 \\ \underset{1 \le k \le C}{\operatorname{argmin}} \frac{\|D(\mu_k, z) - m_k\|}{s_k}, & \text{otherwise} \end{cases} \tag{6}$$

IV. Experiments

We evaluate the proposed self-supervised pre-training method with two types of downstream loss functions: triplet loss [20] (representation loss) and cross-entropy loss (classification loss). Moreover, we test the proposed approach on two malware datasets:

Microsoft Challenge (MC) [21] contains disassembled malware samples from 9 families:"Ramnit", "Lollipop", "Kelihos ver3", "Vundo", "Simda", "Tracur", "Kelihos ver1", "Obfuscator.ACY " and "Gatak". We use 10260 samples that can be correctly parsed then extracted their FCGs as in [5] for the experiment. To simulate an open-world dataset, we randomly pick six classes of digits as the known classes participant in the training, while the rest are considered as unknowns that only exist in the test set.

Android Genome (AG) consists of 1,113 benign android apps and 1,200 malicious android apps. The benign samples are provided by our colleague, and the malicious samples are from [22]. We select nine families with a relatively larger size for the experiment to be fairly split into the training and test sets. The nine families contain 986 samples in total. We first use [23] to extract the function instructions and then generated the FCGs as in [5]. Also, to simulate an open-world scenario as the MC dataset, we randomly pick six digits as the known classes in the training set while considering the rest as the unknown class, which are only used in the test phase.

A. Experimental Setup

As described in Section III, our proposed approach first extracts the FCGs from the malware samples, then uses self-supervised DTAE [2] for pre-training before applying downstream fine-tuning tasks. We experiment with classification loss (cross-entropy loss: ce) and representation loss (triplet loss: triplet) as loss functions in the fine-tuning network for the OSR tasks. To demonstrate that our proposed approach is effective for OSR problems, we compare our approach with OpenMax [1]. Moreover, to prove that the self-supervised pre-training step benefits the OSR tasks, we compare the results of using and not using self-supervised pre-training for the two types of loss functions mentioned above.

As illustrated in Figure 2a, the pre-trained network contains an encoder and a decoder. Furthermore, the learned encoder is fine-tuned with downstream OSR tasks. For the encoder, the padded input layer is of size (67,67) for both MC and AG datasets. Two non-linear convolutional layers follow the input layer with 32 and 64 nodes. We apply the max-pooling layers with kernel size (3, 3) and strides (2, 2) as well as batch normalization after each convolutional layer. After a convolutional block, we add one fully connected non-linear layer with 256 hidden units before the representation layer, containing six dimensions. Moreover, We use the Relu activation function and set the Dropout's keep probability as 0.2. We use Adam optimizer with a 0.001 learning rate. The decoder in the pre-trained network is simply the reverse of the encoder in our experiments. The encoder and representation layer maintain the same architecture and hyperparameters in the fine-tuning network. Meanwhile, the decoder is replaced with different fully connected layers associated with different loss functions.

B. Evaluation Criteria

To simulate an open-set scenario, we randomly pick six out of nine classes as the known classes and used them in training, and samples from the other classes are regarded as the unknown class, which only exists in the test set. We simulate three different open set groups for each dataset and then repeat each group 10 runs, so each dataset has 30 runs. We calculate the average results of 30 runs for performance evaluation.

We perform a three-dimensional comparison for our proposed approach. First, to show that our proposed approach can achieve good performance in the OSR problem, we compare our proposed approach with the popular OSR solution OpenMax [1]. Moreover, to verify that the self-supervised pre-training process benefits the OSR problem for different downstream loss functions, we compare the model performances with and without using the pre-training process. Finally, we compare our proposed transformation methods "FCG-shift" and "FCG-random" with other graph transformations "Node dropping" (ND) and "Subgraph sampling" (SS), which are generally beneficial across datasets [12]. While the AUC score under 100% FPR is commonly used in model performance measurements, the AUC score under 10% FPR is more meaningful for malware detection applications. Moreover, we measure the F1 scores for classifying the known classes correctly and recognizing the unknown class correctly for the OSR system. Finally, to show that our proposed statistical approach to recognizing unknown classes in Section III-E performs as good as the manual thresholding approach: sort the outlier score of the training date in ascending order and then manually pick an outlier score value (99 percentile) as the outlier threshold as in [2], [7], [8], we compare two different thresholding strategies – "manual threshold" and "statistical threshold" – on the representations learned by the vanilla models without pre-training process. To verify that our proposed approaches achieve significant improvement on the OSR, we perform t-tests against OpenMax with 95% confidence in both the AUC scores and F1 scores.

C. Experimental Results

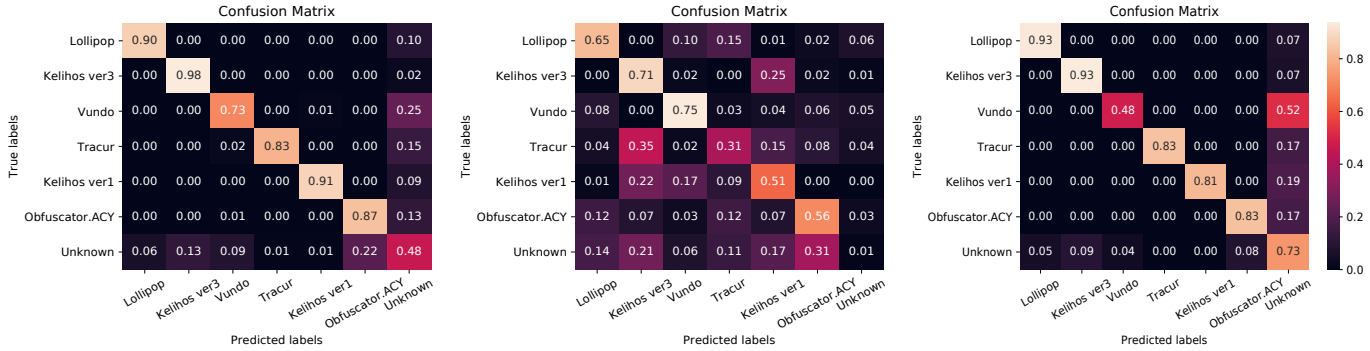
We test our proposed pre-training strategy on downstream networks with classification (cross-entropy loss) and representation (triplet loss) loss functions and apply the statistical thresholding approach to learned representations. Table II shows the average ROC AUC scores of the model performances in two malware datasets under different FPR values: 100% and 10%. Comparing "ce" and "triplet" columns with "OpenMax" columns, we observe that no matter with or without our proposed pre-training process, the models that use cross-entropy loss and triplet loss perform better than OpenMax for our malware datasets. Furthermore, our proposed pre-training approach outperforms the models without the pre-training process in all 8 cases (2 datasets $\times$ 2 FPRs $\times$ 2 loss functions). On the contrary, the DTAE pre-training with node dropping transformation does not benefit the model performance, and the subgraph sampling transformation even hurts the model performance. For MC dataset, the FCG-random transformation works better than the FCG-shift transformation. Meanwhile, their performances differ with different loss functions for the AG dataset.

Table II: The average AUC scores of 30 runs at 100% and 10% FPR of OpenMax and a group of 5 methods for each of the two types of loss functions (ce and triplet): without pre-training, pre-training via DTAE with transformations node dropping (ND), Subgraph sampling (SS), FCG-shift and FCG-random. The values in bold are the highest values in each group. The underlined values show statistically significant improvements (t-test with 95% confidence) comparing with OpenMax.

		OpenMax	ce	triplet
	FPR		No pre-training / ND / SS / FCG-shift (ours) / FCG-random (ours)	No pre-training / ND / SS / FCG-shift (ours) / FCG-random (ours)
MC	100%	0.880±0.037	0.918±0.036 / 0.914±0.063 / 0.626±0.054 / 0.938±0.015 / **0.947±0.011**	0.929±0.020 / 0.919±0.032 / 0.723±0.071 / 0.932±0.017 / **0.933±0.015**
	10%	0.040±0.003	0.053±0.008 / 0.053±0.014 / 0.018±0.005 / 0.061±0.003 / **0.063±0.003**	0.058±0.004 / 0.056±0.006 / 0.036±0.008 / **0.061±0.003** / **0.061±0.003**
AG	100%	0.457±0.200	0.852±0.056 / 0.820±0.128 / 0.418±0.080 / **0.865±0.060** / 0.854±0.062	0.868±0.046 / 0.818±0.124 / 0.427±0.094 / 0.873±0.036 / **0.883±0.035**
	10%	0.001±0.001	0.021±0.012 / 0.019±0.016 / 0.002±0.002 / **0.022±0.013** / 0.019±0.009	0.024±0.010 / 0.018±0.011 / 0.002±0.002 / 0.025±0.011 / **0.027±0.011**

Table III: The average F1 scores of 30 runs of OpenMax and a group of 6 methods (without pre-training using manually selected threshold as baseline, without pre-training using statistical threshold, pre-training via DTAE with transformations node dropping, subgraph sampling, FCG-shift and FCG-random) for each of the two types of loss functions (ce and triplet). The values in bold are the highest values in each group. The underlined values are statistical significant better than OpenMax.

		MC			AG		
		Known	Unknown	Overall	Known	Unknown	Overall
OpenMax		0.891±0.006	0.737±0.010	0.869±0.006	0.408±0.190	0.640±0.163	0.441±0.184
ce	No pre-training (manual threshold)	**0.899±0.010**	0.703±0.061	0.871±0.017	0.683±0.117	0.540±0.329	0.663±0.120
	No pre-training (statistical threshold)	0.890±0.021	0.663±0.176	0.858±0.042	0.705±0.088	0.512±0.363	0.678±0.120
	Node dropping	0.852±0.077	0.715±0.097	0.833±0.078	0.684±0.176	0.636±0.339	0.677±0.181
	Subgraph sampling	0.000±0.000	0.384±0.000	0.055±0.000	0.006±0.018	0.616±0.210	0.093±0.016
	FCG-shift (ours)	0.896±0.010	0.765±0.024	0.878±0.011	**0.743±0.088**	**0.612±0.327**	**0.724±0.113**
	FCG-random (ours)	0.898±0.012	**0.774±0.025**	**0.880±0.013**	0.647±0.129	0.608±0.318	0.641±0.127
triplet	No pre-training (manual threshold)	0.905±0.007	0.728±0.035	0.879±0.011	0.753±0.074	0.789±0.133	0.758±0.068
	No pre-training (statistical threshold)	0.903±0.010	0.749±0.036	0.881±0.013	0.771±0.059	**0.827±0.093**	0.779±0.054
	Node dropping	0.884±0.036	0.736±0.046	0.862±0.037	0.679±0.184	0.768±0.170	0.692±0.171
	Subgraph sampling	0.014±0.075	0.372±0.069	0.065±0.054	0.011±0.061	0.657±0.135	0.104±0.036
	FCG-shift (ours)	**0.906±0.007**	0.758±0.021	**0.885±0.008**	0.745±0.074	0.744±0.250	0.745±0.092
	FCG-random (ours)	**0.906±0.007**	**0.763±0.020**	**0.885±0.008**	**0.776±0.061**	0.819±0.166	**0.782±0.067**

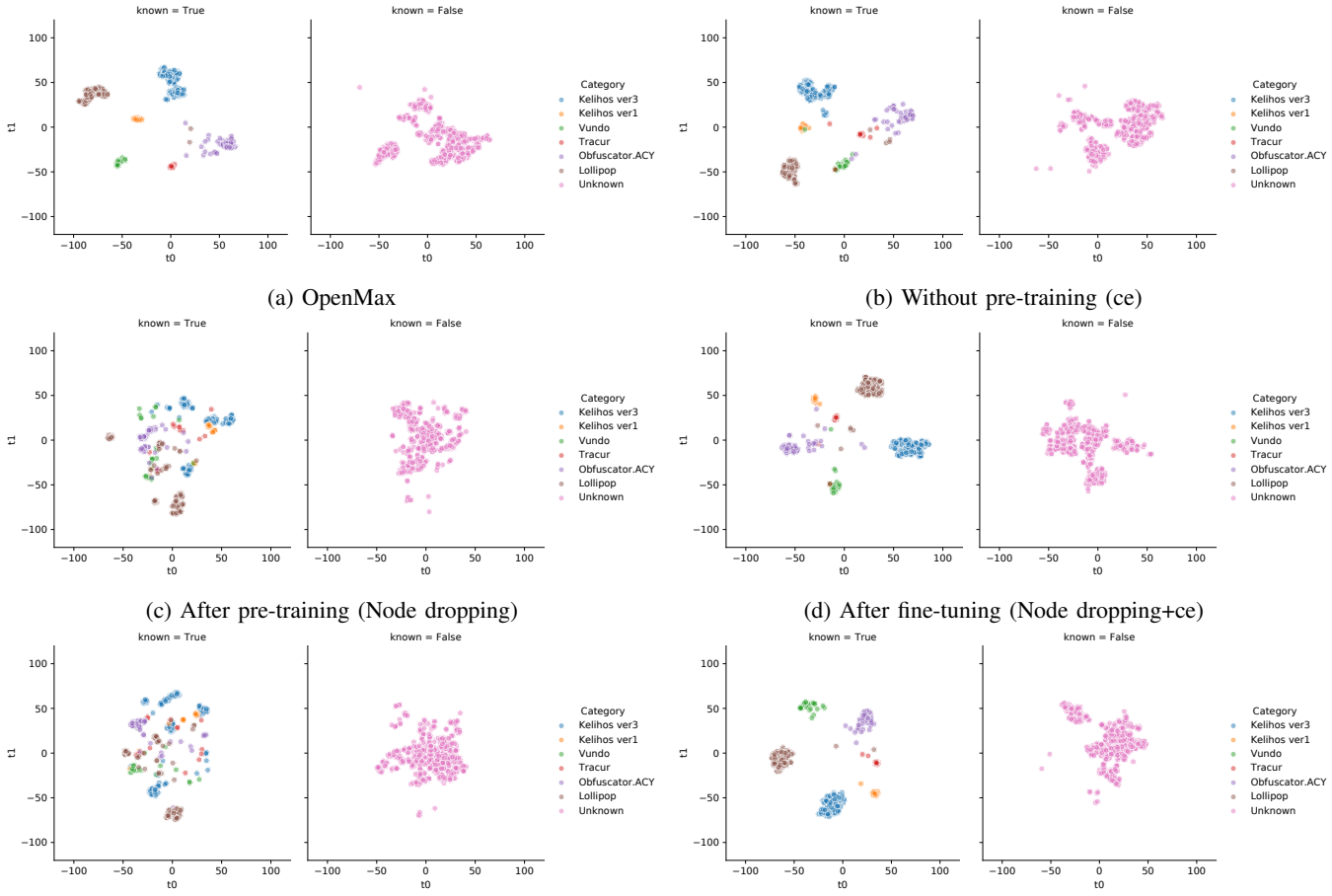


(a) Without pre-training (ce) (b) After pre-training (FCG-random) (c) After fine-tuning (FCG-random+ce)

Figure 3: The confusion matirces of the MC test dataset under different settings: (a) OpenMax; (b) Augmented with node dropping and pre-trained with DTAE; (c) Fine-tuned with cross-entropy loss after (b); (d) Cross-entropy loss without pre-training; (e) Augmented with FCG-random and pre-trained with DTAE; (f) Fine-tuned with cross-entropy loss after (e).

We also measure the OSR performances via F1 scores under different categories. As shown in Table III. The three categories are: "Known", "Unknown", and "Overall". Specifically, the "Known" category is the average F1 scores of the known classes. Moreover, the "Overall" category is the average F1 scores of the known and unknown classes. We observe that the pre-training with our proposed transformation methods improves the model performances in the majority of the cases. However, the pre-training with node dropping and subgraph sampling hurts the model performance in most cases. Moreover, the results in the "manual threshold" and "statistical threshold" rows indicate that our proposed statistical thresholding strategy in Section III-E can achieve similar performance with the manually selected threshold. Meanwhile, the statistical thresholding approach reduces the number of hyperparameters and alleviates the grid searching process.

Overall, we notice that for both ROC AUC scores and F1 scores, the DTAE pre-training using our proposed transformation approach benefits the model performance in OSR problems. Meanwhile, the transformation method node dropping



(a) OpenMax

(b) Without pre-training (ce)

(c) After pre-training (Node dropping)

(d) After fine-tuning (Node dropping+ce)

(e) After pre-training (FCG-random)

(f) After fine-tuning (FCG-random+ce)

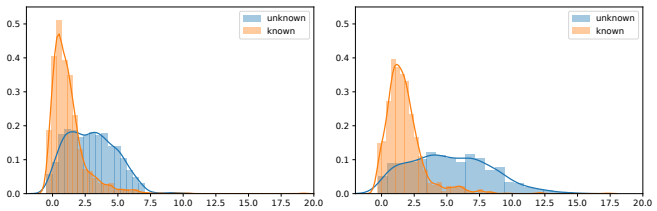
Figure 4: The t-SNE plots of the MC test representations learned by different settings: (a) OpenMax; (b) Cross-entropy loss without pre-training; (c) Augmented with node dropping and pre-trained with DTAE; (d) fine-tuned with cross-entropy loss after (c); (d) Augmented with FCG-random and pre-trained with DTAE; (e): fine-tuned with cross-entropy loss after (d). The left subplots are the representations of the known class, and the right subplots are the representations of the unknown classes.

does not help malware FCGs datasets. As discussed in Section III-C, the FCGs are, in general, very sparse graphs. Dropping nodes and subgraph sampling will potentially lose important information about the malware. Meanwhile, our proposed FCG-shift and FCG-random transformation will preserve all the information by creating isomorphic views.

D. Analysis

While the ROC AUC and F1 scores show that our proposed pre-training approach improves the models' performances, we plot the confusion matrices of one set of the experiments with the MC test set to analyze the experiment results further. In the experiments, the known malware classes are "Lollipop", "Kelihos ver3", "Vundo", "Tracur", "Kelihos ver1", and "Obfuscator.ACY", the remaining three classes together are considered as the unknown class not participating in the training process. Figure 3a shows the confusion matrix of the model using cross-entropy without pre-training. Figure 3b and Figure 3c are the confusion matrices of the model performance after pre-training with FCG-random and after being fine-tuned with cross-entropy loss, respectively. According to the true positive (TP) predictions along the diagonals of the confusion matrices in Figure 3b, the model can already classify the known classes after the pre-training stage. Comparing the model performance without pre-training in Figure 3a and the one with pre-training in 3c, we observe that the TP predictions have been significantly increased for the unknown class. While the TP predictions on the "Vundo" class have decreased, the False Positive (FP) predictions (off-diagonal values) happen only between the known classes and the unknown class instead of among the known classes, which indicates that the known classes are more separable.

To visualize the differences between learned representations, we generate the t-SNE plots of the representations at different stages in different experiments as in Figure 4. Specifically, Figure 4a is the t-SNE plot of the learned representations of OpenMax. Figure 4b shows the representations learned by the model using cross-entropy loss without pre-training. Figures 4c and 4d are the representations learned by the model after



(a) Without pre-training (ce) (b) With pre-training (FCG-random + ce)

Figure 5: The distributions of outlier scores for the known and unknown classes of the MC dataset using cross-entropy loss with and without pre-training process.

pre-training with node dropping and being fine-tuned with cross-entropy loss. Figure 4e and Figure 4f are the representations learned by pre-trained model using DTAE with FCG-random and after being fine-tuned with cross-entropy loss. From the left subplot in Figure 4c and Figure 4e, we observe that even without class label information, the self-supervised pre-training model can capture some cluster information. We can find the tiny clusters for the "Obfuscator.ACY" class, "Kelihos ver3" class and "Lollipop" class, which explains the behavior in Figure 4c and Figure 3b. Moreover, in Figure 4f, the representations of the known classes in the left subplots are more separate from each other. Meanwhile, the representations of the unknown class are more concentrated near the origin.

Figure 5 shows the distributions of the average outlier scores for the known and unknown classes for the MC test set. Comparing the distributions of outlier scores generated from cross-entropy loss without pre-training in Figure 5a and with pre-training in Figure 5b, we notice that while the pre-training process increases the outlier scores for both the known classes and the unknown class, it increases the outlier scores in the unknown classes more significantly, which pushes the distribution further away from the known classes. Therefore, there is less overlap and higher accuracy.

V. Conclusion

In this paper, we design a two-stage learning process for learning the representations of the malware FCGs to resolve the set recognition problem of malware samples. Specifically, we propose two transformation methods for the FCGs to facilitate the detransformation autoencoder (DTAE) in the pre-training step. Then, we fine-tune the network with different types of loss functions. Moreover, to find the optimal threshold for the OSR problem, we design a statistical thresholding approach based on the distribution of learned representations. The proposed approach reduced the number of hyper-parameters and hence the costs of the resources for the hyperparameter tuning process. We evaluate the pre-training approach with classification loss and representation loss functions on two malware datasets. The results indicate that our proposed approach can improve both model performances for the OSR tasks.

References

[1] A. Bendale and T. E. Boult, "Towards open set deep networks," in *Proc. of the IEEE conf. on computer vision and pattern recognition*, 2016, pp. 1563–1572.

[2] J. Jia and P. K. Chan, "Self-supervised detransformation autoencoder for representation learning in open set recognition," *CoRR*, vol. abs/2105.13557, 2021.

[3] W. Fan, Y. Ma, Q. Li, Y. He, Y. E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The World Wide Web Conf., WWW 2019*. ACM, pp. 417–426.

[4] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. of the 34th Intl. Conf. on Machine Learning, ICML 2017.*, pp. 1263–1272.

[5] M. Hassen and P. K. Chan, "Scalable function call graph-based malware classification," in *Proc. of the Seventh ACM Conf. on Data and Application Security and Privacy, CODASPY 2017*, pp. 239–248.

[6] B. G. Ryder, "Constructing the call graph of a program," *IEEE Trans. Software Eng.*, vol. 5, no. 3, pp. 216–226, 1979.

[7] M. Hassen and P. K. Chan, "Learning a neural-network-based representation for open set recognition," in *Proc. of the 2020 SIAM Intl. Conf. on Data Mining, SDM, USA*. SIAM, pp. 154–162.

[8] J. Jia and P. K. Chan, "MMF: A loss extension for feature learning in open set recognition," in *ICANN 2021 - 30th Intl. Conf. on Artificial Neural Networks, Proc., Part II*, vol. 12892. Springer, pp. 319–331.

[9] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," in *6th Intl. Conf. on Learning Representations, ICLR 2018*.

[10] T. Chen, S. Kornblith, M. Norouzi, and G. E. Hinton, "A simple framework for contrastive learning of visual representations," in *Proc. of the 37th Intl. Conf. on Machine Learning, ICML 2020*, pp. 1597–1607.

[11] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow twins: Self-supervised learning via redundancy reduction," in *Proc. of the 38th Intl. Conf. on Machine Learning, ICML 2021, Virtual Event*, vol. 139. PMLR, pp. 12 310–12 320.

[12] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," in *Annual Conf. on Neural Information Processing Systems, NeurIPS 2020, virtual*.

[13] M. Jin, Y. Zheng, Y. Li, C. Gong, C. Zhou, and S. Pan, "Multi-scale contrastive siamese networks for self-supervised graph representation learning," in *Proc. of the Thirtieth Intl. Joint Conf. on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada*, pp. 1477–1483.

[14] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *2001 IEEE Symposium on Security and Privacy*. IEEE Computer Society, pp. 38–49.

[15] X. Hu, K. G. Shin, S. Bhatkar, and K. Griffin, "Mutantx-s: Scalable malware clustering based on static features," in *2013 USENIX Annual Technical Conf.* USENIX Assoc., pp. 187–198.

[16] N. M. Kriege and P. Mutzel, "Subgraph matching kernels for attributed graphs," in *Proc. of the 29th Intl. Conf. on Machine Learning, Edinburgh, Scotland, UK*. icml.cc / Omnipress, 2012.

[17] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H. Kriegel, "Protein function prediction via graph kernels," in *Proc. Thirteenth Intl. Conf. on Intelligent Systems for Molecular Biology 2005, Detroit, MI, USA*, pp. 47–56.

[18] P. Yanardag and S. V. N. Vishwanathan, "Deep graph kernels," in *Proc. of the 21th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, Australia, 2015*, pp. 1365–1374.

[19] GRAND-Lab, "Grand-lab/graph_datasets: A repository of benchmark graph datasets for graph classification (31 graph datasets in total)." [Online]. Available: https://github.com/GRAND-Lab/graph_datasets

[20] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *IEEE Conf. on Computer Vision and Pattern Recognition, CVPR 2015*. IEEE Computer Society, pp. 815–823.

[21] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database," 1999. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[22] Y. Zhou and X. Jiang, "Android malware genome project," 2015. [Online]. Available: http://www.malgenomeproject.org/

[23] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck, "Structural detection of android malware using embedded call graphs," in *AISec'13, Proc. of the 2013 ACM Workshop on Artificial Intelligence and Security, Co-located with CCS*, pp. 45–54.