Unsupervised Graph Contrastive Learning with Data Augmentation for Malware Classification

Yun Gao
Graduate School of Informatics
Nagoya University
Nagoya, Japan
email:gaoyun@net.itc.nagoya-u.ac.jp

Hirokazu Hasegawa
Center for Strategic Cyber Resilience
Research and Development
National Institute of Informatics
Tokyo, Japan
email:hasegawa@nii.ac.jp

Yukiko Yamaguchi
Information Technology Center
Nagoya University
Nagoya, Japan
email:yamaguchi@itc.nagoya-u.ac.jp

Hajime Shimada
Information Technology Center
Nagoya University
Nagoya, Japan
email:shimada@itc.nagoya-u.ac.jp

***Abstract*—Traditional malware detection methods struggle to quickly and effectively keep up with the massive amount of newly created malware. Based on the features of samples, machine learning is a promising method for the detection and classification of large-scale, newly created malware. The current research trend uses machine-learning technologies to rapidly and accurately learn newly created malware. In this paper, we propose a malware classification framework based on Graph Contrastive Learning (GraphCL) with data augmentation. We first extract the Control-Flow Graph (CFG) from portable executable (PE) files and simultaneously generate node feature vectors from the disassembly code of each basic block through MiniLM, a large-scale pre-trained language model. Then four different data augmentation methods are used to expand the graph data, and the final graph representation is generated by the GraphCL model. These representations can be directly applied to downstream tasks. For our classification task, we use C-Support Vector Classification (SVC) as a classification model. To evaluate our approach, we made a CFG-based malware classification dataset from the PE files of the BODMAS Malware Dataset, which we call the Malware Geometric Multi-Class Dataset (MGD-MULTI), and collected the results. The evaluation results show that our proposal achieved Micro-F1 scores of 0.9975 and Macro-F1 scores of 0.9976. According to our experimental evaluation, the unsupervised learning approach outperformed the supervised learning approach in Graph Neural Networks based on malware classification.**

Keywords*—*malware classification; graph contrastive learning; data augmentation; unsupervised learning

I. Introduction

Fueled by the progress of software technology and the internet's development, thousands of malware are created every day due to the proliferation of malware creation and obfuscation tools. Such a massive flood of data poses a considerable challenge to malware analysts and security response centers (SOCs). Traditional malware detection methods cannot continue to quickly and effectively detect such a massive amount of newly created malware. In past decades, machine learning has played an important role in information security, especially in malware detection and classification tasks. It is also a promising method to detect and classify large-scale newly created malware using the features of samples.

In the field of static malware detection, the feature extraction method of portable executable (PE) files used in the Endgame Malware Benchmark for Research (EMBER) dataset [1] has been widely applied. This feature extraction method directly provides consistent feature vectors to researchers, allowing individuals in the same field to compare their respective proposed methods. The information related to software structure, such as the Control-Flow Graph (CFG), is rarely extracted, and most methods are based on surface analysis for extracting statistical information as features. In addition, in most malware detection and classification scenarios, the model is supervised for end-to-end training.

Supervised learning requires manual labeling of a large amount of data, and the model effect depends on the quality of the labels. Therefore, the future research trend, which is exploring unsupervised learning methods, is critical for malware detection and classification. In recent years, Graph Neural Networks (GNNs) have made remarkable progress. We can exploit their powerful representation ability to better represent malware and improve the effectiveness of its detection and classification. However, one remaining difficulty is how to represent malware in graphical form. Since CFG is a natural graph structure, we can generate the graph structure data of malware by extracting CFG. Therefore, we seek to classify malware by constructing a graph dataset and using unsupervised learning. Since no publicly available graph classification dataset exists for malware classification, we started by creating such a dataset.

Our contributions can be summarized as follows:

- We propose a malware classification framework based on graph contrastive learning under unsupervised learning.
- We retain the structural information of the samples extracted from CFG and embed the text features of each

node with a pre-trained language model.
- We create a special graph dataset for malware classification that can be used directly on GNNs.
- Our pre-trained model can effectively perform a low-dimensional representation of malware with which a variety of downstream tasks can be performed. We have achieved good results on malware family classification tasks.

The remainder of this paper is organized as follows. Section II reviews related researches and highlights their methodological differences. In Section III, we discuss the principles of our proposed data augmented GraphCL-based static malware PE classification system and its application to malware classification. In Section IV, we briefly discuss the implementation details of our proposal. In Section V, we describe the corresponding experiments and evaluate their feasibility as well as the advantages and limitations of our proposal. Finally, we discuss our conclusion and describe future work in Section VI.

II. Related Work

Static malware detection allows a sample to be classified as malicious or benign without executing it. In contrast, dynamic malware detection is based on its runtime behavior and as well as its analysis, including time-dependent system call sequences [2]–[4]. Although static detection is not generally deterministic [5], its advantages are also evident over dynamic detection, which can identify malicious files before the samples are executed. Since 1995, various machine-learning-based methods for static PE malware detection have been proposed [6]–[8].

A. Supervised-learning-based Methods

Saxe used histograms through byte-entropy values as input features and multilayer neural networks for classification [7]. Raff et al. showed that fully connected and recursive networks can be applied to malware detection problems [9]. They also used the raw bytes of PE files and built end-to-end deep learning networks [8]. Chen proposed robust PDF malware classifiers with verifiable robustness properties [10]. Coull explored malware detection byte-based deep neural network models to learn more about malware and examined the learned features at multiple levels of resolution, from individual byte embeddings to the end-to-end analysis of models [11]. Rudd proposed ALOHA, which uses multiple additional optimization objectives to enhance the model, including multi-source malicious/benign loss, count loss on multi-source detections, and semantic malware attribute tag loss [12].

B. Supervised Graph Classification

Graph classification assigns a label to each graph to map the graph to the vector space. A graph kernel is dominant in history. It uses the kernel function to measure the similarity between graph pairs and maps graphs to a vector space with a mapping function. In the context of graph classification, GNNs often employ readout operations to obtain a compact representation at the graph level. GNNs have attracted a lot of attention and demonstrated amazing results in this task.

The Dynamic Graph Convolutional Neural Network [13] (DGCNN) uses K nearest neighbors (KNN), builds a subgraph for each node based on the node's features, and applies a graph convolution to the reconstructed graph. The Graph Isomorphism Network (GIN) [14] presents a GIN that adjusts the weights of the central nodes by learning, theoretically analyzes the GIN's expressiveness better than such GNN structures as the Graph Convolutional Network (GCN), and achieves state-of-the-art accuracy on multiple tasks.

C. Unsupervised Graph Classification

Graph2vec [15] uses a set of all the rooted subgraphs around each node as its vocabulary through a skip-gram training process. Infograph [16] applies contrastive learning to graph learning, which is carried out in an unsupervised manner by maximizing the mutual information between graph-level and node-level representations.

Recently, contrast learning has received much attention. It has also been applied in the field of malware detection and classification. Yang presented a novel system called CADE, which can detect drifting samples that deviate from existing classes, and explained detected drift [17]. EVOLIoT [18] is a novel approach that combats "concept drift" and the limitations of inter-family IoT malware classification by detecting drifting IoT malware families and examining their diverse evolutionary trajectories. This robust and effective contrastive method learns and compares semantically meaningful representations of IoT malware binaries and codes without expensive target labels.

III. Proposed Data Augmented GraphCL-Based Static Malware PE Classification

Our proposal is a data augmented GraphCL-based static malware PE classification framework, which can obtain a graph-level representation from malware. We directly extract malware CFG from PE files and through graph contrastive learning obtain a representation of the malware with a vector notation. Finally, malware representations can be performed downstream for various tasks. Graph-level representation shows good performance on malware classification tasks. Next we scrutinize the framework.

A. Raw Graph Generation

To train the GNNs, we need to produce graph datasets, and the main task of this module is to convert PE files into raw graphs. The overview of raw graph generation is shown in Fig. 1.

1) CFG Structure and Disassembly Code: First, the CFG information is extracted from the original PE file samples, the structure information of the basic blocks is retained, and the disassembly code of each basic block is extracted. Each basic block of CFG has a corresponding disassembly code, and the relationship between each basic block is directional. Disassembly codes need to be transformed into feature vectors of specific dimensions to train GNNs. Since the malware CFG

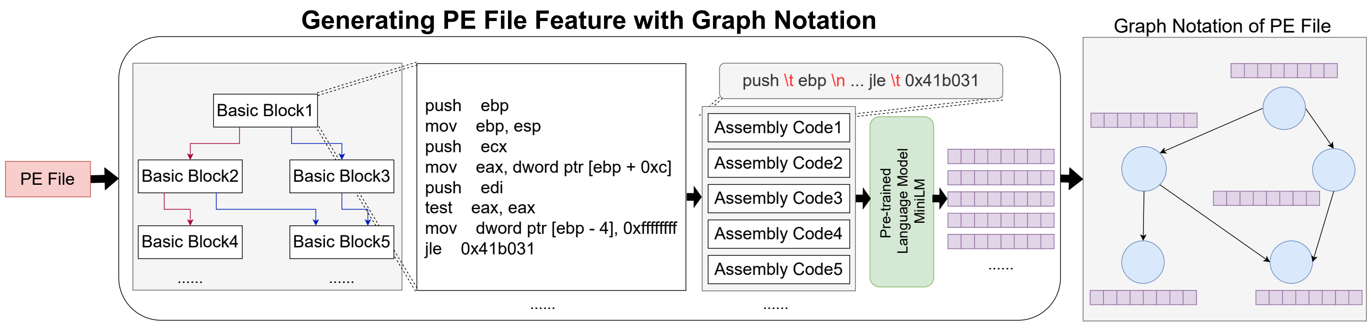


Figure 1. Raw graph generation for proposal

is usually a very large graph, extracting the CFG is very time consuming. Since the disassembly code in each basic block of CFG contains rich semantic information, we need to completely exploit that information and suitably embed it, for example, using a large pre-trained language model.

2) Pre-trained Language Model MiniLM: MiniLM is a method released by Microsoft based on reducing large-scale transformer pre-trained models into smaller models [19]. This Deep Self-Attention Distillation (DSAD) method uses large-scale data for pre-training. The model we use is called "all-MiniLM-L12-v2," which has a 1-billion-sized training set and is designed as a general-purpose model. MiniLM model is a 12-layer transformer with a 384 hidden size and 12 attention heads that contain about 33 M parameters. It maps sentences and paragraphs to a 384-dimensional dense vector space and can be used for tasks like clustering or semantic search. This model is the fastest generation of related studies and still provides good quality. In this step, a 384-dimensional dense vector is generated for each CFG node using the pre-trained model. This vector is added to the corresponding nodes of the directed graph to generate complete graph data with node feature vectors. These directed graphs are used as our raw graph data.

B. Data Augmentation for Graphs

We used the following four data augmentation methods. As shown in Fig. 2, our proposal uses two of them. The best combination is explored in Section 5.

1) Node Dropping: Randomly discard some parts of the vertex and its connections. The missing parts of the vertices do not affect the semantic meaning of the graph, and so the learned representation is consistent under the disturbance of nodes. The dropping probability of each node follows a default Bernoulli uniform distribution (or any other distribution).

2) Edge Perturbation: Randomly add or remove a certain ratio of edges so that the learned representation is consistent under edge perturbation. The prior information of the representation is that adding or removing some edges does not affect the semantics of the graph. The dropping probability of each node follows a default Bernoulli uniform distribution. We only used Edge Removing in this evaluation.

3) Attribute Masking: Randomly removing the attribute information of some nodes motivates the model to use other information to reconstruct the masked node attributes. The masking probability of each node feature dimension follows a default uniform distribution. We only used simple Feature Masking.

4) Subgraph Sampling: Use random walk subgraph sampling [20] to extract subgraphs from the original graph. The basic assumption is that a graph's semantic information can be preserved in its local structure.

Table I overviews the data augmentation for graphs. The default augmentation (dropping, perturbation, masking) ratio is set to 0.1, and the walk length is set to 10.

TABLE I. OVERVIEW OF DATA AUGMENTATION FOR GRAPHS

Data Augmentation	Type	Default Setting
Node Dropping	Nodes, edges	Bernoulli distribution (ratio = 0.1)
Edge Perturbation	Edges	Bernoulli distribution (ratio = 0.1)
Attribute Masking	Nodes	Uniform distribution (ratio = 0.1)
Subgraph Sampling	Nodes, edges	Random Walk (length = 10)

C. Graph Contrastive Learning

Motivated by recent developments in graph contrast learning, we propose a graph contrast learning framework for malware classification. As shown in Fig. 2, in graph contrast learning, pre-training is performed by maximizing the agreement between two augmented views of the same graph by contrast loss in the potential space. The framework consists of the following four main components:

1) Graph Data Augmentation: Throughout the GraphCL framework, given graph data G, two related augmented graphs, $\hat{G}_i, \hat{G}_j$, are generated as positive sample pairs by data augmentation.

2) GIN-based Encoder: GIN-based encoder $f(\cdot)$ is used to generate graph-level vector representation. There are three layers in the GIN-based encoder, and the hidden layer has 64 dimensions. Through the readout function, the embedding of all the nodes is summed to obtain initial graph representation h_i, h_j for augmented graphs $\hat{G}_i, \hat{G}_j$. Graph contrast learning does not apply any constraint to the GIN-based encoder.

3) Projection Head: Nonlinear transformation $g(\cdot)$, called a projection head, maps the augmented representations to another latent space. Contrast loss is computed in the latent space, and z_i, z_j are obtained by applying a two-layer perceptron (MLP).

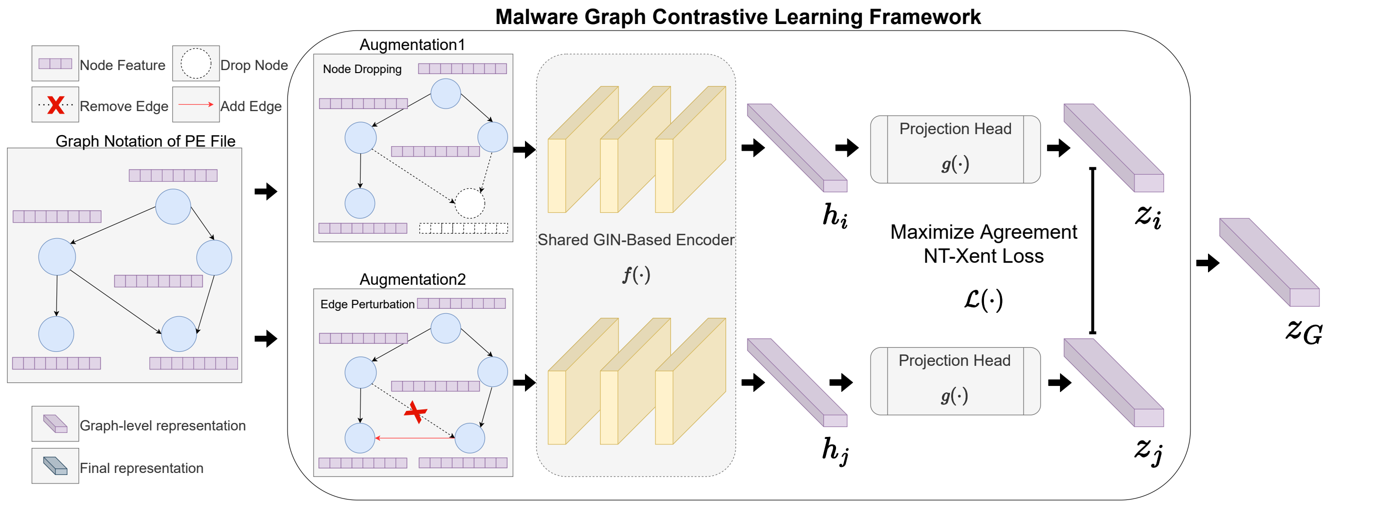


Figure 2. Proposed malware graph contrastive learning framework for graph representation generation

4) Contrastive Loss Function: Contrastive loss function $\mathcal{L}(\cdot)$ is defined to enforce the maximum consistency between positive pairs z_i, z_j and negative pairs. Here we exploit the normalized temperature-scale cross-entropy loss (NT-Xent) [21] [22] and obtain a graph-level final representation of z_G.

D. Graph Classification

By pre-training with GraphCL, we can obtain a valid graph representation z_G. To further verify the effectiveness of our method, different classification models can be chosen for the process, such as random forest, logistic regression, SVM, etc. We chose C-Support Vector Classification (SVC) as the algorithm to validate our pre-trained model's effectiveness.

IV. Implementation Details

We verified the effectiveness of our proposed contrastive learning framework by implementing it with open-source libraries. The implementation details are introduced in this section.

A. Malware Geometric Multi-Class Dataset

1) PE Files Source: Our PE file sample was obtained from the BODMAS Malware Dataset [23]. The software types of all the PE file samples used in our dataset are executable files under an x86-architecture Windows platform without any Dynamic Link Library (DLL) type.

2) Dataset Description: From the BODMAS dataset, we selected eight families of malware and took 500 samples from each family, for a total of 4000 samples in our dataset. Our dataset is named MGD-MULTI. The malware family distribution information is shown in Table II.

Due to the difficulty of collecting benign samples and the imbalanced data problem, we did not include white samples in our multi-class dataset. In our previous malware detection work [24], the MGD-BINARY dataset contained benign samples. We used almost the same GIN model to represent the PE samples, with a slightly different operation of the READOUT layer this time compared to the GIN model in our previous work, giving the final representation a higher vector dimensionality. Based on our previous research, we believe that the GIN model can effectively distinguish benign samples from malicious ones. In future work, we will add benign samples to our dataset.

TABLE II. MALWARE FAMILY DISTRIBUTION OF MGD-MULTI

Family Name	Category Name	Origin Count	Selected Count	Graph Data Size
sfone	worm	4729	500	3.2 GB
upatre	trojan	3901	500	879.4MB
wabot	backdoor	3673	500	4.1 GB
benjamin	worm	1071	500	263.1MB
musecador	trojan	1054	500	1.5 GB
padodor	backdoor	655	500	2.9 GB
gandcrab	ransomware	617	500	6.6 GB
dinwod	dropper	509	500	3.3 GB
Total	-	16209	4000	22.7 GB

Among the different types of malware, we chose families that are more common and have a relatively large number in BODMAS. Due to some limitations of the CFG extraction tool for the PE files we used, many samples couldn't be recognized, causing extraction failure. In addition, for large PE file samples, the process of extracting CFG is very time-consuming. Since the extraction of some samples will fail, we selected a family with more than 500 samples in BODMAS and relatively small original PE files. We further improved the efficiency by only selecting successful samples whose total extraction time is less than 20 seconds in which the total extraction time includes the time of the feature vectors generated by the pre-trained language model. We finally got our MGD-MULTI whose extracted graph data statistical information is shown in Table III.

TABLE III. GRAPH STATISTICS OF MGD-MULTI

Dataset	# Graphs	#Classes	#Features	Avg. #Nodes	Avg. #Edges
MGD-MULTI	4000	8	384	3861.75	5494.82

3) Dataset Splitting: We split 4000 pieces of data in MGD-MULTI into training, validation, and testing sets of 50%, 20%, and 30%, respectively. Since the results of the validation set and the test are similar, only the test set results are shown.

4) Pre-trained Language Model MiniLM: SentenceTransformers is a python framework for state-of-the-art sentence, text, and image embeddings. The initial work was described in a paper from the Sentence-Bidirectional Encoder Representations from Transformers (Sentence-BERT) [25]. We used the MiniLM model provided by the SentenceTransformers library with the model name, all-MiniLM-L6-v2. The model details used in this paper are shown in Table IV.

TABLE IV. PRE-TRAINED MINILM MODEL DETAILS

Name	**all-MiniLM-L12-v2**
Base Model	microsoft/MiniLM-L12-H384-uncased
Max Sequence Length	256
Dimensions	384
Normalized Embeddings	true
Size	120 MB
Pooling	Mean Pooling
Training Data	1B+ training pairs

5) Graph Contrastive Learning: PyGCL [26] is a PyTorch-based open-source Graph Contrastive Learning (GCL) library, which features modularized GCL components from published papers, standardized evaluation, and experiment management. The batch_size of all the experiments is 128, and the optimizer is Adam with a learning rate of 0.0001.

V. Evaluation and Discussion

In this section, we apply the GraphCL model and discuss the experiment results and limitations of our method.

A. Evaluation Metric

We used the following evaluation metrics to assess the performance of our proposed models:

- **The Micro-averaged F1 score** is defined as the harmonic mean of the precision and recall:

$$MicroF1\text{-}score = 2 \times \frac{\text{Micro-Precision} \times \text{Micro-Recall}}{\text{Micro-Precision} + \text{Micro-Recall}}$$

- **The Macro-averaged F1 score** is defined as the mean of the class-wise/label-wise F1-scores:

$$MacroF1\text{-}score = \frac{1}{N}\sum_{N}^{i=0} F1\text{-}score_i$$

where i is the class/label index and N is the number of classes/labels.

B. Evaluation Results

Next we apply the GraphCL model and discuss the experiment results of our method.

1) Different Data Augmentation Combination Results: We selected five different data augmentation methods: Identical (I), Edge Removing (ER), Node Dropping (ND), Feature Masking (FM), and Random Walk Subgraph (RWS). To compare the different data augmentation approaches on the GraphCL model, we used both data augmentation approaches for the input graph itself (Identical + Identical) as the GraphCL model baseline. We also tried different combinations of data augmentation, such as ER and ND, FM and ND, FM and ER, RWS and ER, RWS and ND, and RWS and FM. The experimental results are shown in Table V. The best two data augmentation combinations were RWS and FM. We obtained the best Micro-F1 (0.9958) and Macro-F1 (0.9959).

TABLE V. DIFFERENT AUGMENTATION COMBINATIONS

Method (+SVC)	Augmentation[1]	Micro-F1	Macro-F1
GraphCL	I + I	0.9883	0.9883
GraphCL	ER + ND	0.9925	0.9924
GraphCL	FM + ND	0.9942	0.9942
GraphCL	FM + ER	0.9942	0.9942
GraphCL	RWS[2]+ ER	0.9950	0.9949
GraphCL	RWS + ND	0.9950	0.9949
GraphCL	RWS + FM	**0.9958**	**0.9959**

[1] Default ratio setting is 0.1.
[2] RWS uses a default walk length setting of 10.

2) Best Combination with Different Ratio Results: In the previous set of experiments, we found that the best data augmentation combination is RWS + FM. Based on this combination, we also investigated the results on different ratios on the FM side, and the FM results on different ratios are shown in Table VI.

TABLE VI. BEST COMBINATION WITH DIFFERENT RATIO RESULTS

Method (+SVC)	Augmentation (Ratio)	Micro F1	Macro F1
GraphCL	RWS[1]+ FM (0.1)	0.9958	0.9959
GraphCL	RWS + FM (0.2)	0.9967	0.9967
GraphCL	RWS + FM (0.3)	**0.9975**	**0.9976**
GraphCL	RWS + FM (0.4)	0.9958	0.9958
GraphCL	RWS + FM (0.5)	0.9942	0.9941

[1] RWS uses a default walk length setting of 10.

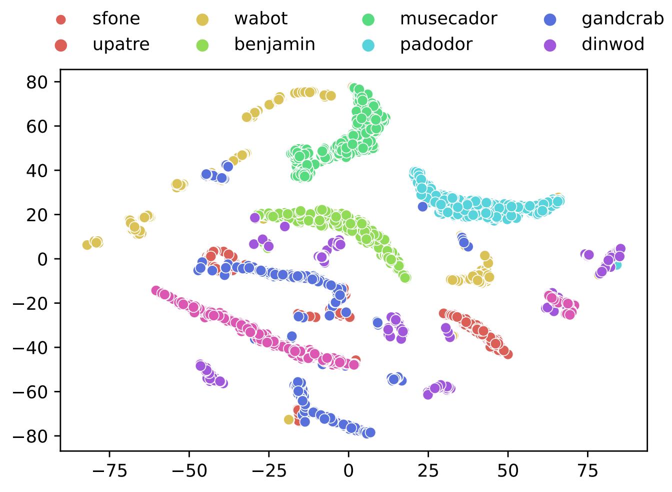
3) Comparison of Different Methods: Our previous studies focused on supervised learning. This study is a graph contrastive learning method in an unsupervised setting. Baseline 1 is a direct graph-level encoding of an input graph using GIN as an encoder, and then the embedding effect is evaluated using SVC. Baseline 2 is data augmentation using the input graph itself. Baseline 3 is our previous work [24] on graph classification, trained using the GIN model in a supervised setting, with a two-layer MLP directly connected after the readout layer for direct classification. A comparison of different methods is shown in Table VII. GraphCL with a setting of RWS + FM (0.3) achieved the best classification results.

TABLE VII. COMPARISON OF DIFFERENT METHODS

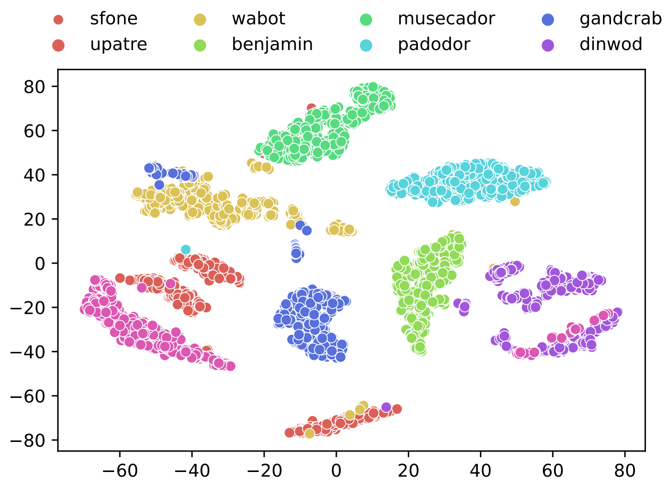
Name	Method	Type	Micro-F1	Macro-F1
Baseline 1	GIN-Encoder + SVC	U[1]	0.9617	0.9620
Baseline 2	GraphCL (I + I) + SVC	U	0.9883	0.9883
Baseline 3	GIN + MLP (Previous work [24])	S[2]	0.9958	0.9957
Proposal	GraphCL (RWS + FM_0.3) + SVC	U	**0.9975**	**0.9976**

[1] U denotes unsupervised learning.
[2] S denotes supervised learning.

We used t-SNE technology to visualize the embedding of Baseline 1 and our proposed method. As shown in Fig. 3, the method of Baseline 1 has already clustered some



(a) Baseline 1: GIN-Encoder + SVC



(b) Proposal: GraphCL (SS+FM_0.3)

Figure 3. t-SNE visualization of Baseline 1 and Proposal

categories, such as the malware of the "padodor" family, but it cannot cluster the "gandcrab" family well. On the other hand, our comparative learning model proposal can better cluster different categories in the eight classes, and a large distance between different categories is maintained.

C. Current Limitations

GraphCL (I + I) is a combination of two Identical, and the effect is equivalent to turning a training set of N samples into 2N samples. The same data model is learned twice for the same data, so the obtained result naturally outperforms GIN-Encoder. The RWS + FM method is most effective because neither method changes the structural information of the original graph. The RWS method samples a subgraph that is smaller than the structure of the original graph, but still retains most of the original graph's structure. For the FM method, the original graph structure is not changed at all, but the values of some dimensions of the node feature vectors are masked, which makes the node features more robust. On the contrary, the other two methods (ER and ND) change the original graph structure more, so the results are lowered.

Because of the relatively large graph structure we extracted from the PE file and the high dimensionality of the nodes in each graph (384 dimensions), our result still leads to a slow training of the GraphCL model even though the dataset size is not too large, only 4000 pieces of data.

The training stage requires around ten minutes with GeForce RTX 3090. We desire a better way to generate node features, such as a lower dimensional in a method that retains its effectiveness.

VI. Conclusion

We proposed the unsupervised learning of different families of malware using graph comparison learning and the multi-classification of learned vectors using SVC and obtained good results. We extracted the CFG of the malware, embedded the disassembly code in a basic block through a large pre-trained language model MiniLM, and obtained a directed graph with node features. The advantage of a directed graph is that it contains the call structure information of the sample in addition to the features of each node. We also produced a multi-classification dataset: MDG-MULTI. Unsupervised GraphCL-based malware classification methods have surpassed graph-based supervised learning methods, such as the Graph Isomorphism Network (GIN) for graph classification. In future work, we will shift our focus to unsupervised learning.

Acknowledgments

This research was partially supported by MEXT/JSPS KAKENHI, Grant Numbers JP19H04108 and 19K11961, and financially supported by JST SPRING, Grant Number JPMJSP2125. We also thank the "Interdisciplinary Frontier Next-Generation Researcher Program of the Tokai Higher Education and Research System."

REFERENCES

[1] H. S. Anderson and P. Roth, "EMBER: an open dataset for training static PE malware machine learning models," *CoRR*, vol. abs/1804.04637, pp. 1–8, 2018.

[2] B. Athiwaratkun and J. W. Stokes, "Malware classification with LSTM and GRU language models and a character-level CNN," *ICASSP 2017*, pp. 2482–2486, 2017.

[3] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," *ICASSP 2013*, pp. 3422–3426, 2013.

[4] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," *ICASSP 2015*, pp. 1916–1920, 2015.

[5] F. Cohen, "Computer Viruses: Theory and Experiments," *Computers & security, Vol. 6, No. 1*, pp. 22–35, 1987.

[6] J. O. Kephart, G. B. Sorkin, W. C. Arnold, D. M. Chess, G. Tesauro, and S. R. White, "Biologically Inspired Defenses Against Computer Viruses," *IJCAI 1995, Vol. 2*, pp. 985–996, 1995.

[7] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," *MALWARE 2015*, pp. 11–20, 2015.

[8] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware Detection by Eating a Whole EXE," *AAAI Workshops 2018*, pp. 268–276, 2018.

[9] E. Raff, J. Sylvester, and C. Nicholas, "Learning the PE Header, Malware Detection with Minimal Domain Knowledge," *AISec@CCS 2017*, pp. 121–132, 2017.

[10] Y. Chen, S. Wang, D. She, and S. Jana, "On Training Robust PDF Malware Classifiers," *USENIX Security 2020*, pp. 2343–2360, 2020.

[11] S. E. Coull and C. Gardner, "Activation Analysis of a Byte-Based Deep Neural Network for Malware Classification," *SP Workshops 2019*, pp. 21–27, 2019.

[12] E. M. Rudd, F. N. Ducau, C. Wild, K. Berlin, and R. E. Harang, "ALOHA: Auxiliary Loss Optimization for Hypothesis Augmentation," *USENIX Security 2019*, pp. 303–320, 2019.

[13] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic Graph CNN for Learning on Point Clouds," *ACM Transactions on Graphics (TOG), Vol. 38, Issue 5*, pp. 1–12, 2019.

[14] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *Proceedings of the 7th International Conference on Learning Representations (ICLR 2019)*, pp. 1–17, 2019.

[15] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning Distributed Representations of Graphs," *CoRR*, vol. abs/1707.05005, pp. 1–8, 2017.

[16] F. Sun, J. Hoffmann, V. Verma, and J. Tang, "Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization," in *Proceedings of the 8th International Conference on Learning Representations (ICLR 2020)*, pp. 1–16, 2020.

[17] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "CADE: Detecting and Explaining Concept Drift Samples for Security Applications," *USENIX Security 2021*, pp. 2327–2344, 2021.

[18] M. Dib, S. Torabi, E. Bou-Harb, N. Bouguila, and C. Assi, "Evoliot: A self-supervised contrastive learning framework for detecting and characterizing evolving iot malware variants," in *Proceedings of ASIA CCS '22: ACM Asia Conference on Computer and Communications Security*, pp. 452–466, 2022.

[19] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers," *CoRR*, vol. abs/2002.10957, pp. 1–15, 2020.

[20] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 855–864, 2016.

[21] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *CoRR*, vol. abs/1807.03748, pp. 1–13, 2018.

[22] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," *CoRR*, vol. abs/2010.13902, pp. 1–12, 2020.

[23] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, and G. Wang, "BODMAS: An Open Dataset for Learning based Temporal Analysis of PE Malware," *DLS 2021*, pp. 78–84, 2021.

[24] Y. Gao, H. Hasegawa, Y. Yamaguchi, and H. Shimada, "Malware detection using attributed cfg generated by pre-trained language model with graph isomorphism network," in *Proceedings of the 12th IEEE International Workshop on Network Technologies for Security, Administration and Protection (NETSAP 2022)*, pp. 1495–1501, 2022.

[25] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," *CoRR*, vol. abs/1908.10084, pp. 1–11, 2019.

[26] Y. Zhu, Y. Xu, Q. Liu, and S. Wu, "An empirical study of graph contrastive learning," *CoRR*, vol. abs/2109.01116, pp. 1–25, 2021.