FCGAT: Interpretable Malware Classification Method using Function Call Graph and Attention Mechanism

Minami Someya*†, Yuhei Otsubo†* and Akira Otsuka*
*Institute of information Security, Kanagawa, Japan
†National Police Agency, Tokyo, Japan
Email: mgs227501@iisec.ac.jp

***Abstract*—Malware classification facilitates static analysis, which is manually intensive but necessary work to understand the inner workings of unknown malware. Machine learning based approaches have been actively studied and have great potential. However, their drawback is that their models are considered black boxes and are challenging to explain their classification results and thus cannot provide patterns specific to malware. To address this problem, we propose FCGAT, the first malware classification method that provides interpretable classification reasons based on program functions. FCGAT applies natural language processing techniques to create function features and updates them to reflect the calling relationships between functions. Then, it applies attention mechanism to create malware feature by emphasizing the functions that are important for classification with attention weights. FCGAT provides an importance ranking of functions based on attention weights as an explanation. We evaluate the performance of FCGAT on two datasets. The results show that the F1-Scores are 98.15% and 98.18%, which are competitive with the cutting-edge methods. Furthermore, we examine how much the functions emphasized by FCGAT contribute to the classification. Surprisingly, our result show that only top 6 (average per sample) highly-weighted functions yield as much as 70% accuracy. We also show that these functions reflect the characteristics of malware by analyzing them. FCGAT can provide analysts with reliable explanations using a small number of functions. These explanations could bring various benefits, such as improved efficiency in malware analysis and comprehensive malware trend analysis.**

I. Introduction

Malware classification, which identifies malware families or categories, is helpful for efficient malware analysis. Static analysis, which requires the interpretation of binary code, is highly technical and time-consuming. In particular, detailed analysis of unknown malware requires considerable effort. In such cases, identifying malware species allows analysts to infer their behavior. Attackers create numerous malware variants that are sophisticatedly designed to evade traditional signature-based detection. Signature-based methods cannot keep up with creating pattern files and are limited in real-time response to variants and new malware species.

Machine learning-based classification methods have been actively studied to counter the threat of unknown malware. Despite their great potential, their models lack interpretability and have difficulty in explaining the reasons behind malware classification. Thus, analysts often struggle to identify the code that characterizes malware. Generally, there is a trade-off between model interpretability and classification performance [3]. Complex models such as deep learning can improve performance due to their high representation capacity. However, these models are treated as black boxes, and it is difficult to obtain explanations for their predictions. Nevertheless, the explanations for incorrect predictions cannot be trusted. Therefore, it is necessary to achieve both high performance and interpretability.

Attention mechanism [2] has recently brought various breakthroughs in the field of machine learning [24]. It computes the weighted average of input features using the learning parameters, attention weights. Throughout this paper, we use the term "*important* feature/function" to denote a feature/function that is assigned a higher attention weight. It means a feature/function that the model considers important for classification. By focusing on important features for prediction, high-performance prediction is achieved. Furthermore, it is expected to obtain explanations for the predictions by extracting the important features. Our method incorporates attention mechanism and tries to achieve both high-performance classification and interpretable explanations. In other words, we try to extract reliable explanations for malware classifications and provide features that characterize the classification classes.

When applying machine learning, it is necessary to determine a feature set to be used. Several granular features can be designed to represent malware, including bytes, basic blocks, and functions. The explanations for classification results are extracted in feature units. If the granularity of the feature is small, such as a byte or basic block, people have difficulty understanding the explanations. Meanwhile, function is relatively superior in interpretability. When analyzing malware, analysts often focus on its function and relevance. Therefore, if the importance of each function is presented, they will easily interpret and utilize it for the analysis. In addition, malware developers rarely create programs from scratch, and some functions are often used interchangeably. Therefore, we adopt functions as features in this study.

Graph Neural Network (GNN) was first proposed by Scarselli et al. [17] and has attracted attention in various fields, such as molecular and human relationships. It has also been applied in malware classification in the past few years and achieved high classification performance [30], [27]. GNN classifies graphs using the graph structures that represent the

Workshop on Binary Analysis Research (BAR) 2023
3 March 2023, San Diego, CA, USA
ISBN 1-891562-84-3
https://dx.doi.org/10.14722/bar.2023.23005
www.ndss-symposium.org

relationships between objects. *Function Call Graph* (FCG), which represents the calling relationship between functions, can be used to represent a program as a graph structure. Wu et al. [27] proposed GEMAL, a malware classifier using FCG structure. They showed that GEMAL can classify malware with high performance by evaluation experiments. In this study, we use GNN to classify FCG in order to achieve high-performance classification.

Considering these backgrounds, we propose an interpretable malware classification method named FCGAT. It classifies malware by focusing on important functions for classification and presents these functions as explanations. FCGAT updates function features reflecting the calling relationships between functions using FCG. Then, by incorporating attention mechanism, FCGAT achieves both high-performance classification and interpretability.

The contributions of this study are as follows:

- We propose FCGAT, a novel malware classification method that can explain the classification result on a per-function basis. FCGAT provides analysts with an importance ranking of function, which is expected to be effective in reducing the burden of following in-depth analysis. To the best of our knowledge, this is the first study to explain malware classification on a per-function basis.
- We evaluate the performance of FCGAT on two datasets and compare it to existing methods. The results show FCGAT succeeded in classification with F1-Score 98.15% and 98.45%, which is comparable to the cutting-edge methods. These results support the effectiveness of malware classification using function features.
- We examine how much the important functions contribute to the classification. Surprisingly, only top 6 (average per sample) important functions yield as much as 70% accuracy. This result suggests malware can be analyzed by focusing on fewer features.
- We analyze the explanations provided by FCGAT and obtain insight into the functions that characterize malware. As an example, we analyzed ransomware samples and identified AES encryption and decryption functions. The insights from these explanations could be helpful for malware analysis and comprehensive malware trend analysis.

This paper is organized as follows. Section II surveys the works related to machine learning-based malware classification. Section III describes the prior knowledge required for this paper. In Section IV, our method is proposed. Our experimental results are given in Section V. Section VI discusses our approach. In Section VII, conclusions are provided.

II. Related Work

Malware analysis methods can be broadly classified into static analysis and dynamic analysis. Our method is classified as a static analysis that does not run malware. Among them, this study is related to the malware classification method. In addition, we target malware in *Portable Execution* (PE) format, which is an executable file format for Windows. Ma et al. [8] have demonstrated nine existing studies on the family classification of PE malware using four different datasets. In this study, we compare our experimental results with those of Ma et al. There are three main types of existing approaches: image-based, binary-based, and disassembly-based, and our proposed method is classified as a disassembly-based method. The following subsections mainly focus on the methods used for comparison in this study.

A. Image-based methods

Image-based methods convert binary malware files into grayscale or color images and then adopt image classification models for malware classification. This method was first proposed by Nataraj et al. [12]. Ma et al. used four different image classification models in their comparison experiments (ResNet-50 [5], VGG-16 [19], Inception-V3 [22] and IMCFN [23]). Image-based methods can use existing image classification models and are compatible with machine learning. However, since we aim to explain the classification reasons, explanations by image pixels that are difficult to interpret are not appropriate.

B. Binary-based methods

Binary-based methods input the malware binaries and process them as time-series data. Raff et al. [15] proposed MalConv, the first end-to-end malware detection model to take the entire malware executable file as input. Qiao et al. [14] proposed a method combining Word2Vec [11] and Multi-Layer Perceptron (MLP). This method first preprocesses to remove meaningless bytes, and then the 256-byte embedding vectors obtained by Word2Vec are input to the MLP classifier. The binary-based approach has the advantage of low preprocessing costs because it uses byte sequences in the executable file. However, the problem is that adjacent bytes are not always contextually meaningful since jump and function call instructions hop to a distanced address. Therefore, our method uses FCG to provide a contextually meaningful representation.

C. Disassembly-based methods

In disassembly-based methods, features are created from assembly code obtained by disassembling an executable file. Our proposed method is classified as this approach. MAGIC by Yan et al. [30] uses DGCNN [32], a type of GNN, to classify *Control Flow Graph* (CFG) with the basic blocks as nodes. MAGIC uses manually designed features of basic blocks. Awad et al. [1] treat an opcode sequence as a document and apply Word2vec to obtain a vector representation of malware. The k-Nearest Neighbor method is used for classification. In MCSC by Ni et al. [13], opcode sequences are converted to vectors based on SimHash [9], and the imaged vectors are classified using a convolutional neural network. Disassembly-based methods have lower performance than the other methods in the Ma et al. experimental results. In MalwareBazaar dataset, the average F1-Score for image-based methods is 96.65%, and binary-based is 97.1%, whereas disassembly-based is 91.98%, which is inferior to the other methods. We aim to build a high-performance malware classifier using function features, even with disassembly-based methods.

Recently, Wu et al. [27] proposed GEMAL, a malware classification method based on FCG. GEMAL uses Word2vec to convert instructions into vectors and then adds the instruction vectors in a function to generate an embedded vector of the function. For graph vectorization, GEMAL uses a graph embedding network based on an attention mechanism, inspired by Massarelli et al. [10]. GEMAL successfully classified malware families with 99.81% accuracy on Microsoft Malware Classification Challenge dataset BIG-2015 [16]. This result is a high performance compared to other existing studies. Inspired by GEMAL we propose a new interpretable malware classification method using FCG. In Wu et al. work, the interpretability of GEMAL is not discussed. Since the source code of GEMAL is not publicly available, we implemented GEMAL based on the original paper [27] and the source code [1] by Massarelli et al. [10] from which GEMAL was derived. We then conduct a replication experiment and compare the results with our method.

D. Explanation techniques in malware classification

Yakura et al. [29] proposed a method to extract malware family-specific regions by using an image classification model with an attention mechanism. The input data are images scaled to 64 x 64 resolution, and the outputs are attention maps indicating important regions for classification. The attention maps are expected to be useful for malware analysis because it extracts byte sequences that characterize malware families. According to evaluation experiments, the performance of malware family classification has an error rate of 50.97%, and it is stated in the paper that there is room for improvement in classification accuracy. Therefore, we aim to create an interpretable model with higher classification performance.

In the latest study, Herath et al. [6] proposed CFGExplainer, an explanatory method in GNN-based malware classification models. CFGExplainer obtains the node importance in CFG subgraph by pruning nodes that do not contribute to classification and identifying the subgraphs that contribute the most to classification. Since CFGExplainer uses CFG, it does not apply to our target FCG-based method. Herath et al. analyzed malware behavior from a subgraph that is 20% of the total CFG, but it is a daunting task to analyze a subgraph with probably more than several thousand basic blocks. We aim to make analysis more efficient by using FCG, which has fewer nodes than CFG, to identify functions that are typical of malware.

III. Preliminary

This section describes the prior knowledge required for this paper.

A. Graph neural network (GNN)

GNN [17] have recently attracted attention in machine learning and are expected to be applied to a wide range of areas represented by graph structures, including networks, human relationships, and molecules [28]. GNN can perform several tasks, such as node classification, graph classification, and link prediction. In this study, we perform graph classification of FCGs in which program functions are nodes and functions in a calling relationship are connected by edges.

[1]https://github.com/lucamassarelli/Unsupervised-Features-Learning-For-Binary-Similarity

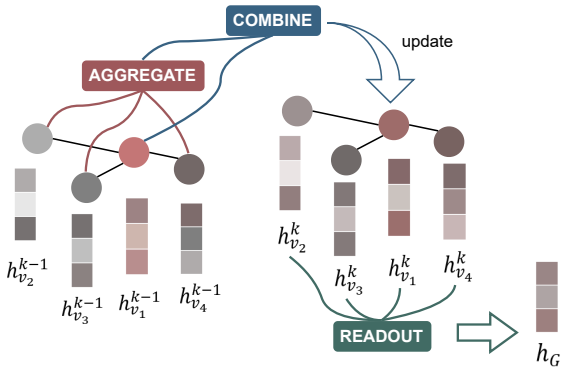

Fig. 1. Overview of graph classification process in GNN. This figure shows the process of updating the features of the node v_1.

1) Graph definitions: A directed graph $G = (V, E)$ is represented by a set of nodes $V = \{v_1, ..., v_n\}$ and a set of edges $E = \{e_1, ..., e_n\}$. Each $e = (u, v) \in E$ is an edge from a node u to v. For each node v, $\mathcal{N}(v) = \{u \mid (u, v) \in E\}$ represents the set of adjacent nodes. Consider giving features to nodes in this graph. Each node v has an initial feature vector $\mathbf{x}_v \in \mathbb{R}^d$, where d is a dimension of feature space. The total number of nodes in the graph is n, and the feature matrix is $\mathbf{X} \in \mathbb{R}^{n \times d}$. Denote the latent feature vector of a node by $\mathbf{h}_v$ and that of the entire graph by $\mathbf{h}_G$.

2) Graph Classification in GNN: The process of graph classification in GNN is divided into three major steps.

1) AGGREGATE: Aggregate the features of neighbor nodes.
2) COMBINE: Update the features of the node using the features of the aggregated neighbor nodes.
3) READOUT: Obtain a representation of the entire graph from the nodes in the graph.

The kth update formula for the node is represented as

$$\mathbf{h}_v^k \leftarrow \mathrm{COM}\left(\mathbf{h}_v^{k-1}, \mathrm{AGG}\left(\left\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\right\}\right)\right),$$

where COM represents COMBINE function and AGG represents AGGREGARE function. The following explains the example in Fig. 1. The set of neighbor nodes of v_1 is $\{v_2, v_3, v_4\}$. Input their latent feature vectors $\{\mathbf{h}_{v_2}^{k-1}, \mathbf{h}_{v_3}^{k-1}, \mathbf{h}_{v_4}^{k-1}\}$ into the AGGREGATE function. The output and the current feature vector $\mathbf{h}_{v_1}^{k-1}$ are input to the COMBINE function, and the output is updated as a new feature vector $\mathbf{h}_{v_1}^{k}$. This process is repeated for other nodes. Finally, READOUT summarizes the vectors of all nodes to obtain a single vector $\mathbf{h}_G$ that represents the entire graph. This process allows obtaining $\mathbf{h}_G$ even if the number of nodes varies from sample to sample. Various methods have been proposed for READOUT processing, including sum, average, and maximum. Our proposed method uses Set2Set [26] in this process and emphasizes the weighting of important functions with the attention weight.

IV. Proposed Method

An overview of the proposed method FCGAT is shown in Fig. 2. FCGAT consists of three parts: FCG creation, function

feature creation, and malware classification. FCGAT takes a binary file as input. A FCG is created from a binary file, and feature vectors of functions extracted using Word2vec are assigned as the node features in the graph. Finally, FCGAT classifies malware class by GNN using the FCG as input.

A. Creation of Function Call Graph

FCG is a directed graph where nodes represent functions and edges represent function calls. Although a general FCG connects edges from the calling function to the direction of called function, FCGAT reverses this arrow. In programs, the called function can be nested within the calling function. Therefore, the features of the called function should be aggregated in the calling function, thus reverse FCG is used. In a general FCG, the ends of the arrows usually represent API functions (often API wrappers). That is, FCGAT aggregate the features of API functions into the calling functions.

B. Function feature creation

This module converts assembly functions into function vectors that can be handled by machine learning. These function vectors need to reflect the semantics of the functions in order to classify malware with high performance. FCGAT uses CBOW model of Word2vec [11] to obtain instruction vectors corresponding to assembly instructions. This method is based on natural language processing techniques, where instructions correspond to words and functions correspond to sentences. CBOW model obtains instruction vectors by predicting the current instruction from surrounding instructions. Word2vec uses unsupervised learning and does not require any prior knowledge. This method is well suited for learning features of malware functions that are difficult to label, unlike open-source programs.

Since assembly instructions contain numerical values such as memory addresses and offset values, the vocabulary becomes excessively large if used as is. Therefore, the assembly instructions are filtered as follows:

1) Remove "," and comments after ";".
2) Replace the number with "N".
3) Connect operand and opcode with "_".

This filtering makes use of the analysis results of analysis tool. In our implementation, we use IDA Pro as the analysis tool. IDA Pro converts well-known functions, such as API functions, from addresses to function names so that instruction vectors can reflect the characteristics of well-known function names.

After creating the instruction vectors in the CBOW model, FCGAT averages the instruction vectors in the function to obtain a function vector. This vector is assigned as the feature of a node in a FCG.

C. Malware classification model

The structure of the classification model is shown in Fig. 2. The inputs are FCGs with function feature vectors as node features. The FCGs are convolved with *Graph Attention Network* (GAT) [25]. The features of the nodes are updated by the following:

$$\mathbf{u}_i = \mathrm{LeakyReLU}(\mathbf{W_1}\mathbf{x}_{v_i} + \mathrm{LeakyReLU}(\|_{k=1}^{K} \sum_{\mathbf{x}_j \in N(v_i)} \alpha_{ij}^k \mathbf{W}^k \mathbf{x}_j)),$$

where $\mathbf{x}_{v_i}$ is the initial function vector at a node to be updated, $\mathbf{x}_j \in N(v_i)$ is the neighbor of the node, $\mathbf{W}_1$ and $\mathbf{W}^k$ are the learning parameters. Multi-head Attention is used to concatenate the results from K-independent Attention mechanisms. The α_{i_j} represents the importance between nodes. α_{i_j} is learned by the following:

$$\alpha_{ij} = \mathrm{softmax}_j\left(\mathrm{LeakyReLU}\left(\mathbf{a}^\top \left[\mathbf{W}_2\mathbf{x}_{v_i} \| \mathbf{W}_2\mathbf{x}_{v_j}\right]\right)\right).$$

The GAT layer's attention mechanism is not used for interpretation. This is because α_{ij} given for each edge depends on the number of neighbors, making it hard to reflect the importance of a node. For example, if there is only one neighbor, α_{ij} will be the maximum possible value of 1, even if the node is not important. In this layer, Multi-head Attention has the effect of expanding representation, which improves performance.

FCGAT does not convolve the features of the updating node but only aggregates the features of the neighbors. The features of the updating node are converted to the hidden layer dimension at the fully connected layer (corresponding to $\mathbf{W_1}\mathbf{x}_{v_i}$) and added to the output of GAT.

Next, the readout process uses Set2Set [26], which is the key to interpretability, to obtain the feature vector of the entire graph as follows:

$$\begin{aligned}
\mathbf{q}_t &= \mathrm{LSTM}(\mathbf{q}^*_{t-1}) \\
\alpha'_{i,t} &= \mathrm{softmax}(\mathbf{u}_i \cdot \mathbf{q}_t) \\
\mathbf{r}_t &= \sum_{i=1}^{N} \alpha'_{i,t}\mathbf{u}_i \\
\mathbf{q}^*_t &= \mathbf{q}_t \| \mathbf{r}_t.
\end{aligned}$$

Set2Set is an extension of the seq2seq [21] approach to support unordered sets. That is, permuting $\mathbf{u_i}$ and $\mathbf{u}_{i'}$ has no effect on the read vector $\mathbf{r}_t$. The ordering of the nodes in FCG depends on the starting address of the function, but this ordering is usually meaningless. Thus FCG is considered suitable to be processed by Set2Set. Fig. 3 shows an overview of Set2Set process and how it explains the classification results. FCGAT can apply the attention mechanism to the variable number of nodes by using LSTM, and we expect that the features of nodes that are important for classification are given high attention. In other words, we can obtain the important functions by analyzing the value of $\alpha'_{i,t}$. Based on attention weights in the final step of Set2Set, function name and importance $\alpha'_{i,t}$ are provided as an importance ranking. In order to reflect the importance for classification to the attention weight, simple network structures are adopted for the other parts. For example, only one convolution in GAT is used to avoid excessively diluting the original function's features.

Finally, classification is performed in a Fully Connected layer. This layer takes a graph vector $\mathbf{h}_G$ as input and outputs

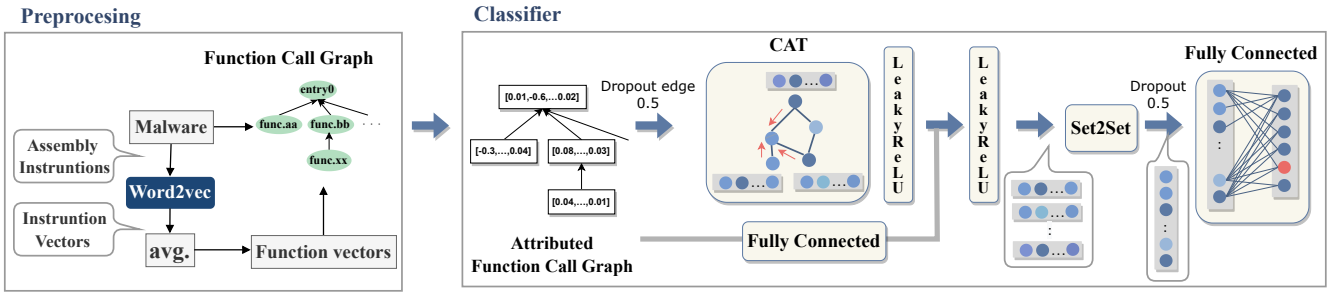

Fig. 2. Overview of our proposed method FCGAT.

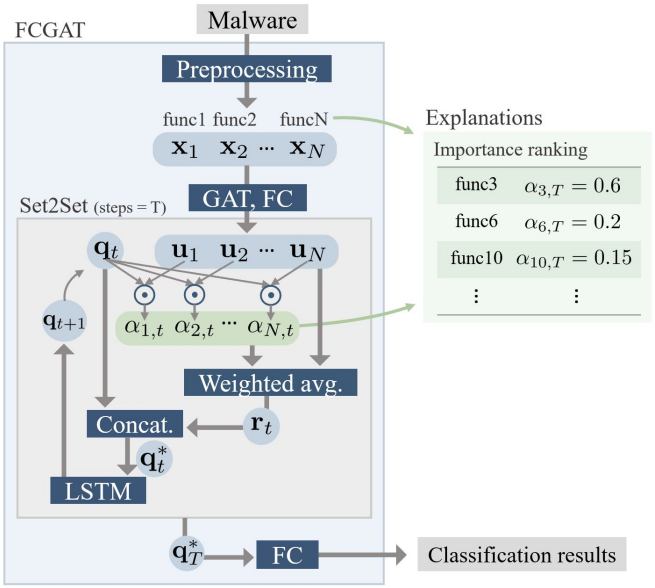

Fig. 3. An overview of Set2Set process and how it explains the classification results.

a vector representing the classification probabilities of the classes as follows:

$$\mathbf{p} = \text{softmax}(\mathbf{W}_3\mathbf{h}_G),$$

where $\mathbf{W}_3$ is a learninig parameter. The index with the highest value among the $\mathbf{p}$ elements indicates the predicted class.

We adopt Dropout [20] to prevent overfitting and stabilize learning. Dropout randomly drops layer outputs to 0 during training to enable correct recognition even if some data are lost. It helps avoid overfitting some local features and improves the robustness of the model. In Fig. 2, Dropout 0.5 means dropping the node's output to 0 with 0.5 probability. In addition, cross-entropy is used as the error function, and AdamW [7] is used as the optimization algorithm.

V. Evaluation

We evaluated FCGAT in terms of classification performance (in Section V-B) and classification interpretability (in Section V-C). Before delving into them, we describe the experimental setup common to both. Our research artifacts are available on Github [2].

[2]https://github.com/som3ya/FCGAT

TABLE I. Classification model parameters

Layer	(In, Out)	Activation	Remark
GAT	(100, 192)	LeakyReLU	K = 3
FC	(100, 192)	LeakyReLU	
Set2Set	(192, 384)		steps = 4
FC	(384, Number of classes)	Softmax	

A. Experimental setup

We implemented FCGAT using Python, PyTorch 1.11.0, and PyTorch Geometric 2.0.2, a library for GNN. FCGAT uses a reverse engineering tool IDA Pro for preprocessing. IDA Python is used to automate operations in CUI, and *ida_gdl.gen_simple_call_chart* function is used to create FCG.

For function feature creation, Gensim library is used to create the CBOW model. When evaluating performance, only train data is used to create CBOW training models, which are then used for the evaluation of test data. We choose parameters embedding size 100, windowsize 2, epochs 100 for the CBOW model. Where, embedding size is the dimension of the feature vector of the function, and window size is the number of surrounding words to be considered. In other words, instruction vectors are learned from the two instructions before and after.

For training the classification model, we choose learning rate 0.001, mini-batch size 256, and epochs 700. The model parameters are shown in TABLE I.

B. Classification performance evaluation

In this experiment, we evaluate the performance of FCGAT on two datasets and compare it with the demonstration results of existing research by Ma et al. [8]. We also compare with GEMAL by Wu et al. [27]. We implemented and performed a replication experiment with GEMAL to compare on the same dataset.

1) Datasets: We used two datasets that Ma et al. used in their experiments on existing studies. We created FCGs and the function vectors for each dataset using FCGAT. Then, the function vectors were assigned to the nodes of the FCG, and datasets with graph format were created. We used the following datasets.

a) MalwareBazaar dataset: MalwareBazaar dataset was published by Ma et al. [8]. MalwareBazaar [3] is a website that provides malware for research purposes. Ma et al. created the dataset using the following procedure. First, they selected

[3]https://bazaar.abuse.ch

TABLE II. MALWAREBAZAAR

Family	Counts
Gozi	767
GuLoader	589
Heodo	214
IcedID	578
njrat	939
Trickbot	881
Total	3,968

TABLE III. BIG-2015

Family	Counts
Ramnit	1,533
Lollipop	2,475
Kelihos_ver3	2,938
Vundo	453
Simda	42
Tracur	751
Kelihos_ver1	387
Obfuscator.ACY	1,217
Gatak	1,012
Total	10,808

the top six malware families from the malware released in 2020 on MalwareBazaar and downloaded 1000 malware samples from each family. Next, they filtered out samples that were not in PE format and used Joe Security [4] and AVClass [18] to check the label of each sample and to determine whether the label removed inconsistent samples. As a result, they obtained 3,971 samples from 6 families. Ma et al. published a list of hash values and family names of the samples on their GitHub [5]. We downloaded 3,971 malware samples using this list and MalwareBazaar's API. We used 3,968 samples (TABLE II) that were successfully preprocessed for the evaluation experiments.

b) BIG-2015: The second dataset is BIG-2015 [16], provided by Microsoft, which consists of 10,868 samples from 9 families. Each sample consists of .byte files, a binary representation of a hexadecimal number, and .asm files, the disassembly outputs of IDA Pro. In this experiment, we created FCGs from .asm files and extracted function binaries. 10,810 samples (TABLE III) that were successfully preprocessed were used in the evaluation experiment.

2) Evaluation metrics: We use the same evaluation metrics as in the experiment of the comparator Ma et al. Four evaluation metrics are used: Accuracy, Precision (P_{macro}), Recall (R_{macro}), and F1-Score ($F1_{\text{macro}}$). Each formula is expressed as following:

$$Accuracy = (\sum_{n=1}^{N} \sum_{m=1}^{M} TP_{mn})/S$$

$$P_{\text{macro}} = \left(\sum_{n=1}^{N} P_n\right)/N$$

$$R_{\text{macro}} = \left(\sum_{n=1}^{N} R_n\right)/N$$

$$F1_{\text{macro}} = \left(\sum_{n=1}^{N} F1_n\right)/N$$

where

$$P_n = \left(\sum_{m=1}^{M} TP_{mn}\right) / \left(\sum_{m=1}^{M} (TP_{mn} + FP_{mn})\right)$$

$$R_n = \left(\sum_{m=1}^{M} TP_{mn}\right) / \left(\sum_{m=1}^{M} (TP_{mn} + FN_{mn})\right)$$

$$F1_n = (2 * P_n * R_n) / (P_n + R_n)$$

N is the number of all classes, S is the number of all samples, and M is the number of cross-validation splits. TP_{mn}, FP_{mn} and FN_{mn} represent the true positive, false positive, and false negative of malware family n in the m-fold.

[4]https://www.joesecurity.org

[5]https://github.com/MHunt-er/Benchmarking-Malware-Family-Classification

3) Experimental results: TABLE IV shows the results of the evaluation metrics computed and compared with the existing studies. For GEMAL, we adopt the results of replication experiments to evaluate performance instead of the paper values because we cannot compare them with MalwareBazaar dataset based on the paper values.

In MalwareBazaar dataset, FCGAT successfully classifies with an F1-Score of 98.15% and outperforms all other methods on all metrics. In BIG-2015, the F1-Score of FCGAT is 98.18%, which is about equivalent to 98.37% in the GEMAL replication experiment.

As noted in Section II-C, the disassembly-based methods perform poorly in the Ma et al. experiments, especially in MalwareBazaar dataset. The reason is considered that these methods do not use the PE header feature of executables. It has been noted that models that take the entire executable file as input, such as MalConv [15], tend to mainly pay attention to the PE header in malware classification [4]. However, classifying malware by focusing on PE headers is risky because PE headers contain information that can be spoofed (e.g., timestamps). FCGAT outperforms the methods that use the entire files, even though it only uses code segment features. We think that FCGAT is able to represent malware features more accurately by extracting features on a function basis.

C. Classification interpretability

In this experiment, we perform malware category classification by FCGAT and extract the importance ranking of the functions as explanations, and verify the effectiveness of these explanations. Malware category indicates malware features such as Downloader, Ransomware, and Trojan. We expect FCGAT to focus on malware features by classifying them into malware categories rather than families. That is, we predict that the important function will be network communication functions for Downloader and encryption functions for Ransomware.

1) Datasets: We used a dataset named BODMAS-8cat, modified from BODMAS [31]. BOSMAS was published by Yang et al. and contains 57,293 executables labeled by 581 families and 14 categories. In this experiment, we excluded as many packed files as possible to focus on the malware features. We excluded from BODMAS those samples that have been pack-detected by PEiD [6]. Then, the category with more than 90 samples in each malware category was selected among the successfully preprocessed samples. As a result, BODMAS-8cat was created, containing 23,369 samples in 8 categories, as shown in TABLE V.

2) Contributions of important functions: In this section, we examine how much the important functions contribute to the classification. Before that, we evaluate FCGAT using BODMAS-8cat with 8:2 hold-out validation. The classification performance of the test data was accuracy of 95.04%. Malware category classification is a more difficult than family classification due to labeling difficulties. However, FCGAT is able to classify with sufficiently high performance, and the explanations for classification are considered reliable.

[6]https://www.aldeid.com/wiki/PEiD

TABLE IV. COMPARISON OF EXPERIMENTAL RESULTS WITH THOSE IN EXISTING STUDIES.

Category	Model	MalwareBazaar dataset				BIG-2015			
		Accuracy	P_{macro}	R_{macro}	$F1_{macro}$	Accuracy	P_{macro}	R_{macro}	$F1_{macro}$
Image	ResNet-50 [5] *	96.68	96.91	96.75	96.83	98.42	96.57	95.68	96.08
	VGG-16 [19] *	96.35	96.58	96.54	96.56	93.94	90.32	81.89	87.27
	Inception-V3 [22] *	95.83	95.67	95.79	95.73	96.99	93.67	94.46	94.03
	IMCFN [23] *	97.38	97.53	97.41	97.47	97.77	95.93	94.81	95.13
Binary	CBOW+MLP [14] *	**97.81**	**97.92**	**98.08**	**98.00**	98.41	97.63	96.67	97.12
	MalConv [15] *	95.92	96.04	96.43	96.20	97.02	94.34	92.62	93.33
Disassembly	MAGIC [30] *	92.82	88.03	87.36	87.45	98.05	96.75	94.03	95.14
	Word2vec+KNN [1] *	95.64	93.34	94.29	93.79	98.07	96.41	96.51	96.45
	MCSC [13] *	96.80	94.97	94.51	94.70	97.94	95.97	96.17	96.06
	FCGAT(proposed method)	**98.11**	**98.03**	**98.27**	**98.15**	**99.27**	**97.93**	**98.45**	**98.18**
	GEMAL(replication) [27]	97.71	97.65	98.00	97.82	**99.37**	**98.26**	**98.48**	**98.37**
	GEMAL(paper) [27] †	-	-	-	-	99.81	-	-	99.81

∗ and † mean the experimental results by Ma et al. [8] and Wu et al. [27], respectively. The top two in each evaluation metric are shown in **bold**. In MalwareBazaar dataset, FCGAT outperformed all other methods on all metrics. In BIG-2015, the F1-Score of FCGAT is 98.18%, which is about equivalent to 98.37% in the replication experiment of GEMAL.

TABLE V. DETAILS OF BODMAS-8CAT.

Category	Family Counts	Sample Counts
backdoor	31	598
downloader	19	967
dropper	17	397
informationstealer	19	347
ransomware	18	169
trojan	282	15,674
virus	8	93
worm	87	5,124
total	481	23,369

To verify that the attention weights reflect the contributing functions for the classification, we evaluate classification performance using only important functions with higher attention weights. This evaluation method is based on the method of Herath et al. [6]. We create subgraph datasets consisting of functions with high attention weights by varying the number of nodes, and evaluate their performance. If FCGAT focuses on functions that highly contribute to the classification, the performance of the subgraph should be close to that of the original graph (100%subgraph).

As mentioned in Section IV-C, we can obtain the important functions by analyzing attention weights in Set2Set. The specific procedures are as follows. First, we create a trained model using all BODMAS-8cat samples. Next, we perform category classification of BODMAS-8cat samples with the trained model and calculate the attention weights corresponding to each function in each sample. We use attention weight $\alpha'_{i,4}$ of the fourth step in Set2Set. Finally, we create subgraphs consisting of nodes from 1 to 100% of the original graph, ordered by attention weight, and calculate the accuracy of the malware categories as predicted by the trained model.

Fig. 4 shows the classification accuracy against Graph Size for the subgraphs, and TABLE VI shows the average number of nodes and classification accuracy for the subgraphs. While the accuracy in the original graph is 95.06%, the 1% subgraph with only 6 nodes on average achieves 69.67%, higher than our intuition. This result supports the claim that FCGAT focuses on functions that highly contribute to classification and implies that only about 6 functions can characterize malware.

In the experiment by Herath et al. [6], the 10% subgraph created by CFGExplainer showed 52.39% accuracy. In contrast, the 10% subgraph created by FCGAT achieves 71.73% accuracy. Note that CFGExplainer uses CFG, which has a different graph structure from our FCG. The nodes in CFG are basic blocks, which are less granular than functions, making it difficult to characterize malware with fewer basic blocks. We believe that FCGAT can reduce the analysis workload by providing the analysts with reliable explanations with fewer functions.

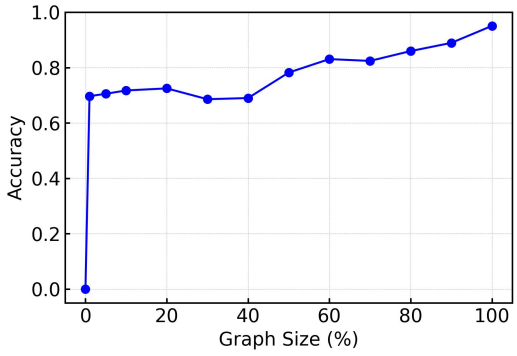

Fig. 4. The classification accuracy of subgraphs. Graph size means the ratio of the number of nodes in the subgraph to those in the original graph. Between 1 and 40% of the graph size, performance does not decrease as the number of nodes is reduced, and even a graph size as small as 1% shows about 70% accuracy.

TABLE VI. AVERAGE NUMBER OF NODES AND CLASSIFICATION ACCURACY OF SUBGRAPHS.

Graph Size (%)	Average Number of Nodes	Accuracy
1	6.2	69.67
5	28.8	70.53
10	56.9	71.73
20	113.4	72.48
30	170.1	68.57
40	226.5	68.99
50	283.0	78.20
60	339.6	83.06
70	396.2	82.40
80	452.7	85.96
90	509.3	88.92
100	565.4	95.06

Even though the average number of nodes in the 1% subgraph is only about 6, it achieves 69.67% accuracy. This means that only about 6 functions can characterize malware.

3) Trend analysis of malware categories: In this section, we analyze the important functions and show that it is possible to analyze trends in the malware category by identifying typical functions for each category.

We extracted the most important functions for each sample by analyzing the attention weights as follows:

1) Create a trained model of FCGAT using all samples in BODMAS-8cat.
2) Perform inference on all samples with the trained model and calculate the attention weight $\alpha'_{i,4}$ of the fourth step in Set2Set.
3) Functions with the max $\alpha'_{i,4}$ are aggregated for each category. In this process, function names generated by IDA Pro are used to determine the function identities. IDA Pro uses the symbol name of the function if symbol data is available. Otherwise, the function name is generated based on the address reference. For *sub_[start address]* and *start*, which are considered to have no correspondence between function name and code, the md5 hash of the function vector is calculated and the function name is changed to *sub_[hash value]* and *start_[hash value]* before aggregation.

In this way, each sample's functions with the max attention weight are summarized for each category, and the top eight are shown in TABLE VII.

The results provide insight into the functions that are typical for each category. Especially in the case of API functions, it is easy to infer behavior from the function name. We will focus on some characteristic functions.

Backdoor is placed on the victim machine by attackers, enabling it to be controlled remotely. Thus, *GetCommandLineA* reflects the feature of reading command line parameters. This function is found in common with the malware families gbot and zegost. The user-defined function *sub_4f847a1* performs XOR operations, which Malware often uses XOR operations to obfuscate malicious data or code.

In downloader, we can see that FCGAT focused on *InternetOpenW* which is used when downloading another malware from the Internet. This result highly reflects the characteristics of downloader.

Dropper is a program that creates and drops files containing malicious code from itself. *CheckSumMappedFile* calculates a new checksum for the file and returns it with the CheckSum parameter. Malware sometimes copies a binary executable to a file in a temporary path, maps the file to memory, and then calls *CheckSumMappedFile* to verify the checksum. *__imp_VirtualProtect* and *ShellExecuteA* are used to execute shell code.

Informationstealer includes the blocker, zbot, and other families.*GetFocus*, typical of this class, gets a handle to the window that has keyboard focus. This function is found in common with blocker.

In trojan, the samples in which *sub_a9f1051* is the most important function are packed. We used PEiD to exclude as many packed samples as possible but could not exclude packers that did not match the PEiD signatures. However, FCGAT captures the characteristics of packers and tends to classify the same packers into the same class. Dealing with packed samples is future work.

We have seen important functions for malware classification. FCGAT seems to identify particularly similar samples within classes and focuses on functions common to them. As a result, we have found many API functions that were often common to multiple samples. By utilizing this method, it will be possible to comprehensively analyze the characteristics of malware categories.

4) Case study: Ransomware/ GandCrab: GandCrab is ransomware that appeared in 2018. Here we will discuss two samples of GandCrab, GandCrab#1 and GandCrab#2, and examine which functions FCGAT focuses on.

The importance ranking of the functions and their attention weights are shown in TABLE VIII. The functions *aes_encrypt* and *aes_decrypt* are at the top of the ranking for both samples. As the name implies, *aes_encrypt* and *aes_decrypt* functions perform encryption and decryption of AES ciphers, and well represent features of ransomware. These functions were named by IDA Pro's Lumina server, which stores metadata for well-known functions.

GandCrab#1 is a DLL file, and GandCrab#2 is an EXE file. Their program structures are quite different. However, they were correctly classified by focusing on the common functions, *aes_encrypt* and *aes_decrypt*. FCGAT seems to be able to capture the characteristics of programs that are semantically similar but structurally and syntactically different. In this case, the function names were named by the Lumina server, so it is easy to guess the behavior from the function names. However, even if it were difficult to guess from the function names (e.g., *sub_420000*), FCGAT can focus on the characteristic functions because it uses features of assembly code. We consider that these explanations can be useful for malware analysis.

VI. Discussion

A. Effectiveness of our method

We will discuss the effectiveness of our proposed method by comparing it with existing studies. In the same way as GEMAL [27], FCGAT uses FCG for malware classification, and both have succeeded in high-performance classification in evaluation experiments (Section V-B3). They differ mainly in the readout process. GEMAL uses a fully connected layer to learn the attention weight and thus requires a constant number of nodes in the input graph. That is, the number of functions must be constant for all samples. GEMAL truncates nodes greater than 200 and zero-padded for fewer. However, the maximum number of nodes in the MalwareBazaar Dataset is 10,789 and in BIG-2015 it is 14,716. For samples with many functions, functions that characterize malware may be truncated. On the other hand, FCGAT adopts Set2Set for the readout process and uses LSTM to achieve an attention mechanism for a variable number of nodes. Therefore, we believe that FCGAT is superior to GEMAL in terms of model interpretability.

As described in Section II-D, Yakura et al.'s method [29] and CFGExplainer [6] have different feature units from our method. Yakura et al. state that the byte sequence of the original executable can be identified by extracting the pixel regions of the image. However, the classified images are compressed to 64 × 64 resolution, and it is suspicious whether

TABLE VII. AGGREGATE RESULTS FOR FUNCTIONS WITH MAX ATTENTION WEIGHTS.

backdoor		downloader		dropper		informationstealer	
CreateFileA	119	InternetOpenW	260	sub_460d8c2	278	GetFocus	202
fclose	105	MessageBoxA	130	GetWindowThreadProcessId	48	mciSendStringA	47
sub_4f847a1	83	mciSendStringA	77	CheckSumMappedFile	17	__dllonexit	11
SetWindowLongA	74	sub_f9da9b9	61	IIDFromString	14	free	10
GetCommandLineA	47	sub_1236153e	48	IsProcessorFeaturePresent	4	CreateFileA	9
sub_a9f1051	29	DispatchMessageA	34	sub_62922e7	3	sub_d0f95e9	8
sub_cf1fee5	18	UpdateWindow	29	__imp_VirtualProtect	2	lstrcatA	7
_fdopen	17	CoUninitialize	27	ShellExecuteA	2	GdipCreateFromHDC	7
ransomware		**trojan**		**virus**		**worm**	
CoRegisterMallocSpy	34	sub_a9f1051	1,316	GetActiveWindow	38	__abnormal_termination	1,314
CoReleaseMarshalData	22	__vbaUI1I4	1,036	strcpy	31	EnterCriticalSection	892
InternetReadFile	19	CloseHandle	709	start_80bc47b	15	sub_f7f9a83	407
ReadFile	12	InternetOpenW	633	?ProcessWndProcException	2	__ZNSt6locale5_ImplC2Ej	255
SetPropA	11	GetSystemDirectoryA	512	nullsub_3	2	__ZNSt6locale5_ImplC1Ej	205
SwitchDesktop	11	sub_acc2cf0	510	CoUninitialize	1	sub_d0f95e9	188
IsProcessorFeaturePresent	10	OleSetMenuDescriptor	472	GetFileVersionInfoSizeW	1	__ZNSt6locale6globalERKS_	182
CreateOleAdviseHolder	7	rtcLowerCaseVar	388	EnableMenuItem	1	SetWindowsHookExA	158

This table shows the most important functions and their counts, aggregated by category. The results provide insights into the functions typical of the malware category.

TABLE VIII. IMPORTANCE RANKING OF FUNCTION.

GandCrab#1		GandCrab#2	
function	$\alpha'_{i,4}$	function	$\alpha'_{i,4}$
aes_encrypt	0.5984	aes_encrypt	0.0331
aes_decrypt	0.4015	aes_decrypt	0.0084
sub_10007BB0	2.002e-06	SetHandleInformation	0.0080
sub_10003F70	4.166e-07	GetTickCount	0.0080
sub_10004C20	7.423e-10	InitializeCriticalSection	0.0080

These samples are common to the top two important functions, *aes_encrypt* and *aes_decrypt*, which are characteristic of ransomware.

the corresponding byte sequence can be identified from the pixel area. In addition, this method had a classification error of 50.97% for malware families, and low classification performance was mentioned as a problem. On the other hand, our method successfully classified malware with high performance, with an F1-Score of over 98%, and both high-performance and interpretable malware classification were achieved.

CFGExplainer obtains explanations for classification in units of basic blocks. As seen in Section V-C2, the classification performance of CFGExplainer is significantly degraded for subgraphs with a small number of nodes because basic blocks are less granular than functions. Therefore, to identify subgraphs that highly contribute to classification, it is necessary to increase the number of nodes in the subgraph, which increases the effort of manual analysis. In this regard, FCGAT can identify subgraphs with fewer nodes and higher contribution to classification, with an average of 6 nodes still performing 69.67% of accuracy.

These existing studies provide analysis cases for individual samples, but they do not analyze to characterize the class of the entire dataset. That is, some samples can provide interpretable explanations for classification, but others may not be well explained. We have confirmed that the important functions reflect the trend of malware categories in Section V-C3. FCGAT extracts the explanations on a per-function basis, which enables aggregating the characteristic functions for each class and identifying trends.

B. Limitations

Similar to existing static analysis methods, our proposed method is challenging to perform malware classification for the malware obfuscated by packers. The packed program contains a decompression processing code for each type of packer. Therefore, it is likely to be able to capture the characteristics of packers and thus be applicable to packer estimation task.

VII. CONCLUSION

We have presented FCGAT, a malware classification method that can explain classifications by functions. Evaluation experiments showed that FCGAT classifies malware with sufficiently high performance compared to the latest methods. We then analyzed the explanations for the classification results based on the attention weight of the classification model. The results show that these explanations provide functions that reflect the malware features. In addition, we performed malware classification on the subgraphs with top important functions. Surprisingly, our result shows that only top 6 (average per sample) of important functions contribute to the classification result with almost 70% of accuracy. Malware families can be characterized by considerably fewer functions than our intuition. FCGAT is expected to improve the efficiency of malware analysis and comprehensive malware trend analysis. For example, it can be combined with existing tools, such as IDA Pro, to highlight functions specific to the malware.

TOOLS

The research artifacts are available at the following URL: https://github.com/som3ya/FCGAT

ACKNOWLEDGMENT

Part of this work was supported by Innovative Science and Technology Initiative for Security Grant Number JPJ004596, ATLA, Japan.

REFERENCES

[1] Y. Awad, M. Nassar, and H. Safa, "Modeling Malware as a Language," *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, May 2018.

[2] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[3] A. Barredo Arrieta, N. Daz-Rodrguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Information Fusion*, vol. 58, pp. 82–115, Jun. 2020.

[4] S. Bose, T. Barao, and X. Liu, "Explaining AI for malware detection: Analysis of mechanisms of MalConv," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Jul. 2020.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, Jun. 2016.

[6] J. D. Herath, P. P. Wakodikar, P. Yang, and G. Yan, "CFGExplainer: Explaining Graph Neural Network-Based Malware Classification from Control Flow Graphs," in *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2022*. IEEE, 2022, pp. 172–184.

[7] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[8] Y. Ma, S. Liu, J. Jiang, G. Chen, and K. Li, "A comprehensive study on learning-based PE malware family classification methods," *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1314–1325, Aug. 2021.

[9] G. S. Manku, A. Jain, and A. Das Sarma, "Detecting near-duplicates for web crawling," in *Proceedings of the 16th international conference on World Wide Web - WWW '07*. Banff, Alberta, Canada: ACM Press, 2007, p. 141.

[10] L. Massarelli, G. A. Di Luna, F. Petroni, L. Querzoni, and R. Baldoni, "Investigating Graph Embedding Neural Networks with Unsupervised Features Extraction for Binary Analysis," *Proceedings 2019 Workshop on Binary Analysis Research*, 2019.

[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Jan. 2013.

[12] 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 - VizSec '11*. Pittsburgh, Pennsylvania: ACM Press, 2011, pp. 1–7.

[13] S. Ni, Q. Qian, and R. Zhang, "Malware identification using visualization images and deep learning," *Computers & Security*, vol. 77, pp. 871–885, Aug. 2018.

[14] Y. Qiao, B. Zhang, and W. Zhang, "Malware Classification Method Based on Word Vector of Bytes and Multilayer Perception," *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, Jun. 2020.

[15] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware Detection by Eating a Whole EXE," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[16] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft Malware Classification Challenge," *ArXiv*, Feb. 2018.

[17] F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini, "The Graph Neural Network Model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, Jan. 2009.

[18] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "AVclass: A Tool for Massive Malware Labeling," vol. 9854. Cham: Springer International Publishing, 2016, pp. 230–253.

[19] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *3rd International Conference on Learning Representations, ICLR 2015*, Sep. 2014.

[20] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 27. Curran Associates, Inc., 2014.

[22] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 2818–2826.

[23] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture," *Comput. netw.*, vol. 171, no. 107138, p. 107138, Apr. 2020.

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017, pp. 5998–6008.

[25] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," *6th International Conference on Learning Representations, ICLR 2018*, 2018.

[26] O. Vinyals, S. Bengio, and M. Kudlur, "Order Matters: Sequence to sequence for sets," 2015.

[27] X.-W. Wu, Y. Wang, Y. Fang, and P. Jia, "Embedding vector generation based on function call graph for effective malware detection and classification," *Neural Computing and Applications*, vol. 34, no. 11, pp. 8643–8656, Jun. 2022.

[28] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[29] H. Yakura, S. Shinozaki, R. Nishimura, Y. Oyama, and J. Sakuma, "Neural Malware Analysis with Attention Mechanism," *Comput. Secur.*, vol. 87, 2019.

[30] J. Yan, G. Yan, and D. Jin, "Classifying Malware Represented as Control Flow Graphs using Deep Graph Convolutional Neural Network," *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 52–63, Jun. 2019.

[31] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, and G. Wang, "BODMAS: An Open Dataset for Learning based Temporal Analysis of PE Malware," in *IEEE Security and Privacy Workshops, SP Workshops 2021*. IEEE, 2021, pp. 78–84.

[32] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *AAAI Conference on Artificial Intelligence*, 2018.