

Interpretable Malware Classification based on Functional Analysis

Miles Q. Li¹ ^a and Benjamin C. M. Fung² ^b

¹*School of Computer Science, McGill University, Montreal, Canada*

²*School of Information Studies, McGill University, Montreal, Canada*

Keywords: Malware Classification, Interpretable Machine Learning, Neural Networks.

Abstract: Malware is the crux of cyber-attacks, especially in the attacks of critical cyber(-physical) infrastructures, such as financial systems, transportation systems, smart grids, etc. Malware classification has caught extensive attention because it can help security personnel to discern the intent and severity of a piece of malware before appropriate actions will be taken to secure a critical cyber infrastructure. Existing machine learning-based malware classification methods have limitations on either their performance or their abilities to interpret the results. In this paper, we propose a novel malware classification model based on functional analysis of malware samples with the interpretability to show the importance of each function to a classification result. Experiment results show that our model outperforms existing state-of-the-art methods in malware family and severity classification and provide meaningful interpretations.

1 INTRODUCTION


In the Internet age, malicious software (malware), as the major means for cyber attacks, has been an increasing threat to legitimate Internet users, financial systems, and government organizations (Ye et al., 2017; Abusitta et al., 2021). It is not uncommon to receive phishing emails, accidentally download adware/ransomware¹, or experience privacy information breach from a giant financial corporation² in recent years. There is thus a pressing need to identify malware and discern its intent and severity before it achieves its nefarious goals.


Manual reverse engineering, as the primary step taken to gain an in-depth understanding of a piece of malware, is a time-consuming process. Due to the increasing volume and complexity of malware, automatic malware analysis techniques have to be applied so as to recognize malware in a timely manner. Signature-based methods have been extensively applied in antivirus engines. Filename, text strings, and regular expressions of byte code etc., can be used as signatures to recognize known malware samples and their non-significant variants. However, they

cannot be used to recognize new malware or significant variants of malware samples with evasive techniques (Abusitta et al., 2021). Machine learning based malware classification methods have been proposed to have better generalizability to identify new malware samples and significant variants.

Different static features have been applied for malware classification, such as PE headers (Baldangombo et al., 2013), byte sequences (Schultz et al., 2001; Saxe and Berlin, 2015), printable strings (Schultz et al., 2001; Saxe and Berlin, 2015), PE imports (Schultz et al., 2001; Saxe and Berlin, 2015), and assembly code (Li et al., 2021b; Moskovitch et al., 2008) etc. Among these methods, Li et al. (2021b) propose to group semantically equivalent assembly code functions into clusters and classify an unknown sample based on the matching between its assembly functions and the known assembly function clusters. Comparing with previous malware classification methods, it has the ability to tell which functions of the unknown sample contribute to the classification result. Their work is then extended through integrating a feedforward neural network as the classification module (Li and Fung, 2022). The extended method yields more accurate classification results for severity classification but loses the interpretability to tell the contribution of each assembly function to the classification result.

When malware analysts analyze a malware sam-

^a  <https://orcid.org/0000-0001-7091-3268>

^b  <https://orcid.org/0000-0001-8423-2906>

¹<https://techcrunch.com/2019/05/12/wannacry-two-years-on/>

²<https://www.upguard.com/blog/biggest-data-breaches>

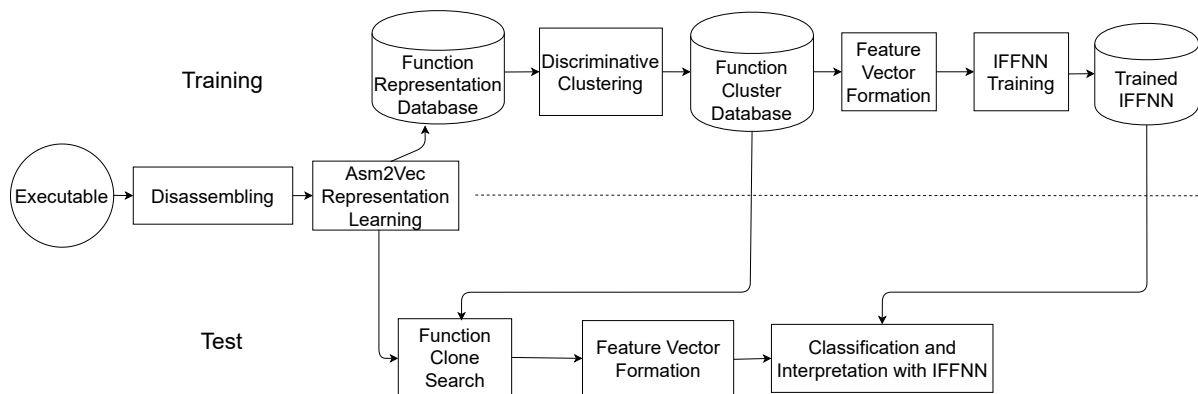


Figure 1: Workflow of our interpretable malware classification model.

ple, they usually do not examine the full set of assembly functions of the sample. Rather, they try to locate and examine only the core functions that are related to the major malicious behaviors so as to discern the intent of the sample or to validate the result of an automatic malware classification system.

In this paper, we improve the aforementioned malware classification method Li and Fung (2022) by integrating an interpretable feedforward neural network (Li et al., 2022) to enhance both the classification performance and interpretability. The proposed classification model is the first that can show the importance/contribution of each assembly function to determine the unknown sample’s class. This interpretation capability can significantly improve malware analysts efficiency in analyzing an unknown sample. Experiment results show that our proposed malware classification model outperforms previous classification methods based on different kinds of static features and provide meaningful interpretations.

We organize this paper as follows. Section 2 discusses related work. Section 3 describe the proposed malware classification model. Section 4 show the experiment setting and results. Section 5 concludes this paper.

2 RELATED WORK

Malware analysis can be conducted dynamically or statically. Dynamic analysis methods require a virtual machine (VM) or simulator to execute the target sample and record its executed assembly instructions (Royal et al., 2006; Anderson et al., 2011), memory image (Dahl et al., 2013; Huang and Stokes, 2016), invoked system calls (Fredrikson et al., 2010), or its high level behaviors (Bayer et al., 2006), such as stealing credentials, key logging, downloading payload etc. The execution process is time-consuming

and dynamic analysis methods can be evaded with an embedded sandbox detector³. The focus of this paper is static methods that do not require any execution of the target sample. The analysis is conducted on the binary content of the sample itself. The common static features extracted from a sample include byte sequences (Schultz et al., 2001; Saxe and Berlin, 2015), assembly code (Li et al., 2021b; Moskovitch et al., 2008), PE header numerical fields (Baldan-gombo et al., 2013), PE imports (Schultz et al., 2001; Saxe and Berlin, 2015), printable strings (Schultz et al., 2001; Saxe and Berlin, 2015), and malware images (Nataraj et al., 2011; Mourtaji et al., 2019) etc. These features can be represented as sequences, vectors, matrices, and graphs. A variety of machine learning models have been applied on those features for malware classification, such as naive Bayes, decision trees, support vector machines, and artificial neural networks.

Among the malware classification studies, the dedicated machine learning model for malware classification proposed by Li et al. (2021b) and then extended with a feedforward neural network (Li and Fung, 2022) is the most related to our proposed method. Their methods are based on grouping semantically equivalent functions in known malware samples into clusters, and classify an unknown malware sample based on the matching between its functions and the known function clusters in different classes. The nature of this kind of methods allows them to show malware analysts the reason for the classification so that they can examine or justify a classification result. The limitation with their methods is that they cannot tell the importance of each function among the tens or hundreds of matched functions to the classification result.

Li et al. (2021a) propose an interpretable feedfor-

³<https://www.ptsecurity.com/ww-en/analytics/antisan-dbox-techniques/>

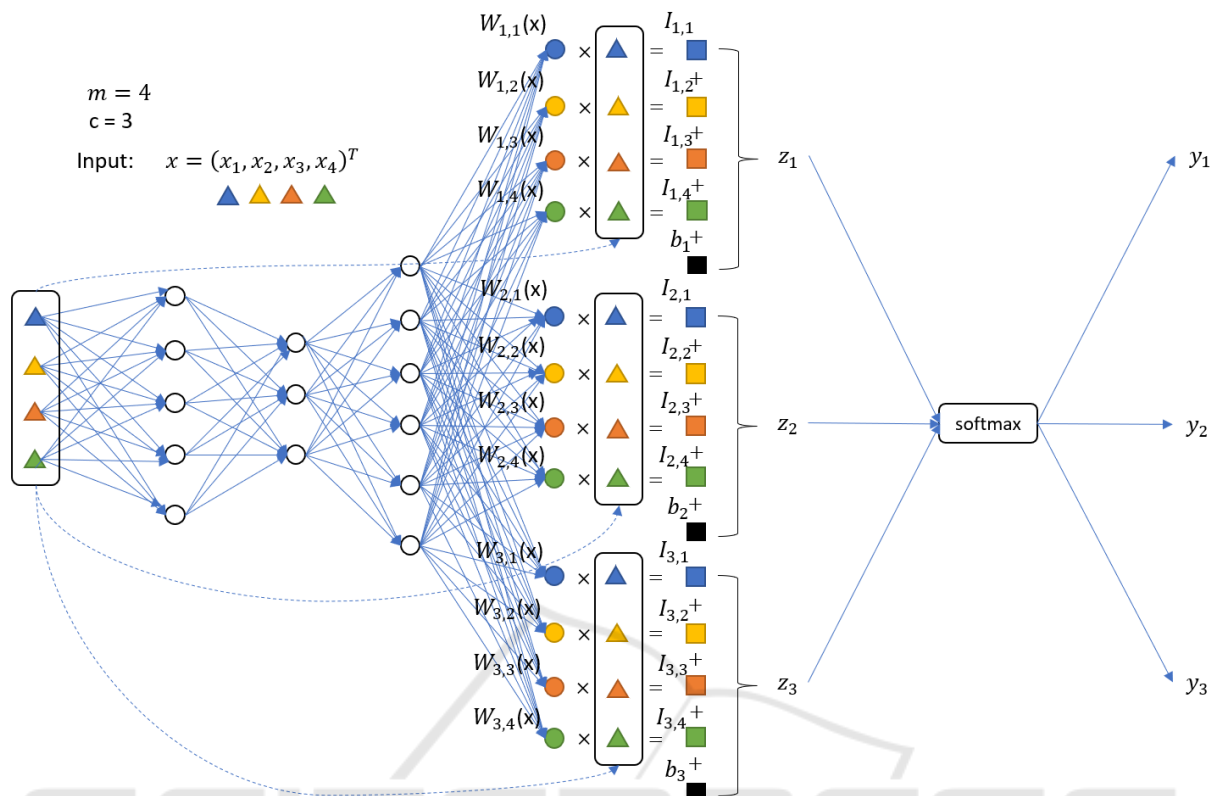


Figure 2: An example to show the IFFNN to classify a sample of dimension 4 and to explain the contribution of each feature to each class. For example, $I_{1,2}$ is the importance of feature 2 to class 1.

ward neural network (IFFNN) for binary classification that can show the importance of each feature to the classification result. The IFFNN is then generalized for multi-class classification and the classification performance and interpretability have been comprehensively evaluated and experiment results show that the classification performance of IFFNNs is similar to their non-interpretable counterparts (i.e., normal feedforward neural networks) and the interpretations are accurate (Li et al., 2022). In this work, we integrate the IFFNN to the dedicated malware classification model (Li et al., 2021b) so that it forms the first interpretable malware classification model that can show the importance of each assembly function to the classification result.

3 METHODOLOGY

Figure 1 presents the workflow of our interpretable malware classification engine for both training and test. We provide the details of the training and test steps in this section.

3.1 Executable Disassembling

As the malware classification model is based on functional analysis, we disassemble every sample for training or test to get its assembly code functions. IDA Pro⁴ and Ghidra⁵ are two commonly used disassemblers to achieve this purpose.

3.2 Assembly Function Representation Learning

For training, we apply Asm2Vec, an assembly function function representation learning method proposed for assembly function clone search, to the assembly functions of all training samples. The training of Asm2Vec and the generation of vector representations for the assembly functions in the training samples happen simultaneously. The vectors representing semantically equivalent assembly functions have a large cosine similarity. For test, we feed the assembly functions of a target sample to the trained Asm2Vec, to generate their vector representations.

⁴<https://www.hex-rays.com/products/ida/>

⁵<https://ghidra-sre.org/>

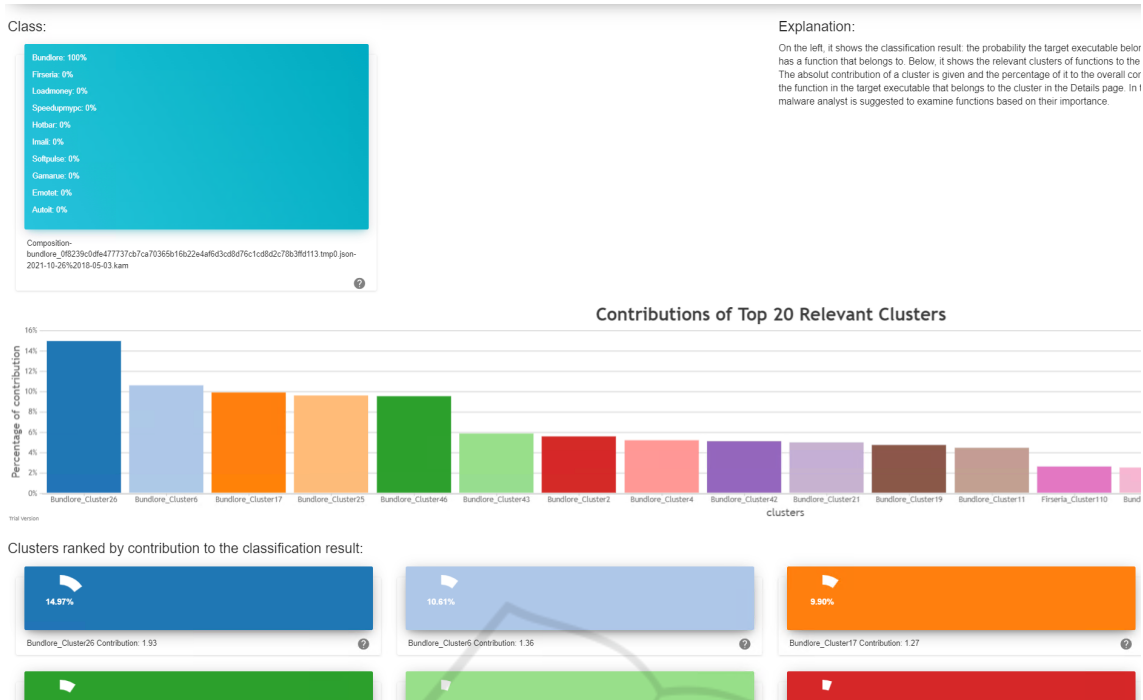


Figure 3: An example of the Summary page of the classification result.

3.3 Discriminative Assembly Function Clustering

In the training process, we then group the semantically equivalent assembly functions into clusters. The assembly functions in each cluster are programmed to achieve the same (or at least similar) purpose. As a clone search engine, Asm2Vec uses the vector representations it generates to determine whether two assembly functions are equivalent to each other. We apply a Union-Find algorithm, namely the Weighted Quick-Union with Path Compression algorithm, to gradually aggregate semantically equivalent assembly functions to a cluster. This algorithm is optimized by locality-sensitive hashing (LSH).

The family of LSH functions (Charikar, 2002) can be described as follows: each hash function corresponds to a random vector r which has the same dimension as the vector representation of an assembly function. The hash value of an assembly function represented as vector u is computed as follows:

$$h_r(u) = \begin{cases} 1 & u \cdot r > 0 \\ 0 & u \cdot r \leq 0 \end{cases}$$

The probability that two assembly functions represented as u and v have the same hash value is as follows:

$$Pr[h_r(u) = h_r(v)] = 1 - \frac{\theta(u, v)}{\pi} \quad (1)$$

By applying a set of n such LSH functions to the representation of an assembly function, we obtain an n -bits signature of it. Assembly functions that are semantically equivalent have a large probability to have the same signature. Therefore, we only apply the Union-Find algorithm to the assembly functions that have the same signatures. We apply l sets of LSH functions to create l signatures for each assembly function to increase the chance that the assembly functions with equivalent semantics to have the same signature at least once so that they can be grouped to the same cluster. We use the way proposed by Li et al. (2021b) to determine the hyper-parameters n and l . We refer the readers to their paper to find the complete details.

After we get the assembly function clusters with the aforementioned method, we need to filter them based on their **discriminative power**. This is because some clusters are the sets of assembly functions that are commonly seen in all classes, and they cannot be used to determine the class of a sample that has a function belongs to them. Hence, they should be filtered. Formally, the discriminative power of a cluster can be defined as follows.

Let $\|comf(G_i, C_j)\|$ be the number of assembly function shared by class C_j and cluster G_i divided by the number of executables in class C_j . The **popularity** of malware class C_j in cluster G_i is defined as fol-

Table 1: Statistics of the datasets.

Malware family classification dataset		Malware severity classification dataset	
Family	Number of Samples	Severity	Number of Samples
Autoit	102	Level 1	237
Bundlore	218	Level 2	1052
Emotet	223	Level 3	175
Fireseria	147	Level 4	109
Gamarue	141	Level 5	128
Hotbar	144	Level 6	64
Imali	136	Level 7	66
Loadmoney	200	Level 8	99
Softpulse	93		
Speedingupmypc	107		
Total	1,511	Total	1,930

lows:

$$pop(G_i, C_j) = \frac{\|comf(G_i, C_j)\|}{\sum_{j=1}^m \|comf(G_i, C_j)\|} \quad (2)$$

The discriminative power of cluster G_i is $dp(G_i) = 0$ if G_i contains only 1 function; and $dp(G_i) = \max_j\{pop(G_i, C_j)\} - \min_j\{pop(G_i, C_j)\}$ otherwise.

We filter clusters of which the discriminative power is lower than a threshold θ and only keep the rest for use in the next steps.

3.4 Assembly Function Clone Search

For a target sample, we use the trained Asm2Vec to determine whether each of its assembly functions is equivalent to an assembly function in one of the discriminative assembly function clusters. This is achieved by comparing the vector representations of the assembly functions of the target sample and the vector representations of the assembly functions in the clusters. If an assembly function of the target sample is semantically equivalent to an assembly function of a cluster, the function belongs to that cluster and the test sample is related to the cluster.

3.5 Feature Vector Formation

For each training sample and test sample, we form a feature vector of dimension m , where m is the total number of discriminative assembly function clusters. The value of each entry is 1 if there is at least one assembly function of the sample belongs to the corresponding cluster; otherwise, the value is 0.

3.6 Classification with Interpretable Feedforward Neural Network

To classify a sample to a malware class, we feed the feature vector formed in the previous step to an interpretable feedforward neural network (IFFNN) proposed by Li et al. (2022). In addition to the classification result, the IFFNN also shows the importance of the clusters for the classification result. An example is shown in Figure 2. The details of the IFFNN are as follows. Let $\mathbf{x} \in R^m$ represent the vector formed in the previous step. It is fed to l fully-connected (FC) hidden layers with *Relu* as the activation function:

$$v_l(\mathbf{x}) = FC^l(\dots FC^1(\mathbf{x})\dots) \quad (3)$$

$$FC^i(\mathbf{v}) = Relu(W_i \mathbf{v} + b_i) \quad (4)$$

$$\mathbf{W}(\mathbf{x}) = Reshape(\mathbf{W}_2 v_l(\mathbf{x}), (c \times m)) + \mathbf{B}_2 \quad (5)$$

$$\mathbf{y} = softmax(\mathbf{W}(\mathbf{x})\mathbf{x} + \mathbf{b}) \quad (6)$$

The contribution/importance of cluster i for classifying the sample to class j is $W(\mathbf{x})_{j,i}x_i$. This is the interpretation for the classification result. The most important clusters and the corresponding functions in the target sample are supposed to be examined by a malware analyst with high priority.

In the training phase, we use the feature vectors of training samples and their class labels to train the IFFNN. We use cross entropy loss as the objective function and Adam (Kingma and Ba, 2014) as the optimizer with the initial learning rate $1e - 4$. In the test phase, we use the trained IFFNN to classify an unknown sample and interpret the result. An example of the user interface to display the interpretation of a classification result is shown in Figure 3.

Table 2: Classification results of different methods on the test sets for malware family classification and severity classification.

Method	Family Classification		Severity Classification	
	Accuracy	P-value	Accuracy	P-value
Mosk2008OpBi	82.9	1.1e-11	78.7	1.0e-12
Bald2013Meta	89.5	3.5e-8	88.2	2.6e-3
Saxe2015Deep	92.8	2.7e-2	88.1	2.1e-4
Mour2019CNN	51.3	2.9e-17	22.8	7.5e-18
Li2021Func	87.1	9.9e-11	70.3	2.9e-15
Li2022Sev	93.2	0.36	89.0	0.22
Our Model	93.5	N/A	89.1	N/A

4 EXPERIMENTS

We conducted experiments to evaluate the proposed interpretable malware classification model in terms of classification performance and interpretability. Following the literature of multi-class classification research, the accuracy of the classification results is used to indicate the classification performance. Accuracy is the ratio of correctly classified samples to all test samples. We use student t-test to indicate statistically significant difference in accuracy between other models and our model.

4.1 Datasets

We evaluate our interpretable malware classification model on malware family classification and severity classification. The malware family dataset contains malicious executables of 10 malware families. The gold standard labels are acquired with the AVClass malware labeling tool (Sebastián et al., 2016) based on analysis reports from VirusTotal⁶. The malware severity dataset contains malicious executables of 8 severity levels that are labelled based on the Kaspersky Lab Threat Level Classification tree⁷. We exclude packed or encrypted samples, so that the assembly functions of the malware samples can be analyzed. We use Pefile⁸ and Yara rules⁹ to detect packing. The statistics of the datasets are given in Table 1. We use k-fold cross-validation where $k = 5$ to evaluate the models. One fold is chosen as the validation set, and one fold is chosen as the test set.

⁶<https://www.virustotal.com/>

⁷<https://encyclopedia.kaspersky.com/knowledge/rules-for-classifying/>

⁸<https://gist.github.com/islem-esi/334d223b3088e0bec5adc75f010c83c2>

⁹https://gist.github.com/islem-esi/cef15f99db844fe1cf596656dfe9bb2#file-detect_packer_cryptor-py

4.2 Malware Classification Methods For Comparison

In the evaluation, we use the following state-of-the-art malware classification methods to compare with our model:

- **Mosk2008OpBi:** Moskovitch et al. (2008) propose to use TF or TF-IDF of opcode bi-grams as features and to use document frequency (DF), information gain ratio, or Fisher score as the criterion for feature selection. They apply Artificial Neural Networks, Decision Trees, Naïve Bayes, Boosted Decision Trees, and Boosted Naïve Bayes as their malware classification models.
- **Bald2013Meta:** Baldangombo et al. (2013) propose to extract multiple raw features from PE headers and to use information gain and calling frequencies for feature selection and PCA for dimension reduction. They apply SVM, J48, and Naïve Bayes as their malware classification models.
- **Saxe2015Deep:** Saxe and Berlin (2015) propose a deep learning model that works on four different features: byte/entropy histogram features, PE import features, string 2D histogram features, and PE metadata numerical features.
- **Mour2019CNN:** Mourtaji et al. (2019) convert malware binaries to grayscale images and apply a convolutional neural network on malware images for malware classification. Their CNN network has two convolutional layers followed by a fully-connected layer.
- **Li2021Func:** Li et al. (2021b) propose to group assembly functions to clusters, and compute the similarity of a query executable to a malware family based on the comparison of the number of clusters of the family related to it and the number of clusters related to a median sample of the training set in the family.

- **Li2022Sev**: Li and Fung (2022) extend **Li2021Func** with a normal feedforward neural network and they show that the new model achieves better accuracy on malware severity classification.

We use grid search to tune the hyper-parameters for all methods on the validation set.

4.3 Classification Performance

Table 2 shows the experiment results of state-of-the-art malware classification methods for comparison and our interpretable malware classification engine. As can be seen, the new interpretable malware classification engine outperforms all other methods in both malware family classification and severity classification. It achieves much higher classification performance than **Li2021Func** and slightly higher performance than **Li2022Sev** in malware family classification and severity classification. The advantage of our model compared to **Li2022Sev** is the interpretability. In many cases, there is a common trade-off between classification performance and interpretability (Arrieta et al., 2020; Rai, 2020). However, the integration of the explainable AI into our engines does not harm the classification performance. In addition, it does not bring much extra computational overhead. This is because the explainable AI module, namely, IFFNN, is intrinsically interpretable, rather than a post-hoc explanation method.

Table 3: The Spearman’s Rank Correlation Coefficients between the feature importance rank given by our novel interpretable malware classification engine, Gini importance (GI), and information gain (IG).

Malware family classification			
	IG	GI	Our Engine
IG	1	0.61	0.58
GI	0.61	1	0.62
Malware severity classification			
	IG	GI	Our Engine
IG	1	0.06	0.19
GI	0.06	1	0.72

4.4 Interpretability

4.4.1 Quantitative Evaluation

We compute the *Gini importance (GI)* and *information gain (IG)* of the clusters for the classification results, and then rank them based on those criteria. We also rank the clusters by their importance for individual predictions. The *i*-th important cluster among all *m* is given a score of $1/m$ for individual predictions.

We then accumulate the scores of the clusters on all test samples. The clusters are then ranked by their accumulated scores. Table 3 shows the Spearman’s Rank Correlation Coefficients between the ranks of these three methods. As can be seen, for malware family classification, the importance ranks of clusters by IG, GI, and our engine have high correlation; for malware severity classification, the ranks given by GI and our engine have higher correlation and they have relatively lower correlation with the rank given by IG. This means that our engine relies more on the clusters that are statistically more important clusters.

4.4.2 Qualitative Evaluation

We also require a malware analyst to manually evaluate the interpretations. He uses the Softpulse and Hotbar malware families for the case study. He first summarizes the most common behaviors of the samples in those malware families. Then, he finds that the most important assembly functions/clusters determined by the engine are indeed part of the characteristic behaviors/patterns of the respective malware families, such as loading malicious DLLs, writing unwanted content to disks, establishing connections to external malicious servers for Softpulse, creating pop-up advertisements, establishing a connection to another socket application, and manipulation of browsers for Hotbar. This result confirms that the interpretations given by our engine are valid.

5 CONCLUSION

In this paper, we propose a novel interpretable malware classification model that can show the importance of each assembly function cluster to the classification result of a sample. It is an improved version of a malware classification model based on functional analysis (Li et al., 2021b). Experiment results show that our classification model outperforms existing state-of-the-art models in terms of classification accuracy and provide meaningful interpretation. Therefore, in addition to identifying the malware class in a timely manner, the proposed malware classification method can also tremendously improve the efficiency of malware analysts in real-world malware analysis scenarios.

ACKNOWLEDGMENT

This research is supported by Defence Research and Development Canada (contract no. W7714-217794),

the Discovery Grants (RGPIN-2018-03872) from the Natural Sciences and Engineering Research Council of Canada, and Canada Research Chairs Program (950-232791).

REFERENCES

- Abusitta, A., Li, M. Q., and Fung, B. C. M. (2021). Malware classification and composition analysis: A survey of recent developments. *Journal of Information Security and Applications (JISA)*, 59(102828):1–17.
- Anderson, B., Quist, D., Neil, J., Storlie, C., and Lane, T. (2011). Graph-based malware detection using dynamic analysis. *Journal in computer Virology*, 7(4):247–258.
- Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., et al. (2020). Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115.
- Baldangombo, U., Jambaljav, N., and Horng, S.-J. (2013). A static malware detection system using data mining methods. *arXiv preprint arXiv:1308.2831*.
- Bayer, U., Moser, A., Kruegel, C., and Kirda, E. (2006). Dynamic analysis of malicious code. *Journal in Computer Virology*, 2(1):67–77.
- Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th annual ACM Symposium on Theory of Computing*, pages 380–388.
- Dahl, G. E., Stokes, J. W., Deng, L., and Yu, D. (2013). Large-scale malware classification using random projections and neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3422–3426. IEEE.
- Fredrikson, M., Jha, S., Christodorescu, M., Sailer, R., and Yan, X. (2010). Synthesizing near-optimal malware specifications from suspicious behaviors. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 45–60. IEEE.
- Huang, W. and Stokes, J. W. (2016). Mtnet: a multi-task neural network for dynamic malware classification. In *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 399–418. Springer.
- Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Li, M. Q., Fung, B. C., Charland, P., and Ding, S. H. (2021a). I-mad: Interpretable malware detector using galaxy transformer. *Computers & Security*, 108:102371.
- Li, M. Q. and Fung, B. C. M. (2022). *Software Technologies*, chapter A novel neural network-based malware severity classification system. Communications in Computer and Information Science (CCIS). Springer.
- Li, M. Q., Fung, B. C. M., and Abusitta, A. (2022). On the effectiveness of interpretable feedforward neural network. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, Padova, Italy. IEEE.
- Li, M. Q., Fung, B. C. M., Charland, P., and Ding, S. H. (2021b). A novel and dedicated machine learning model for malware classification. In *Proceedings of the 16th International Conference on Software Technologies*, pages 617–628.
- Moskovitch, R., Feher, C., Tzachar, N., Berger, E., Gitelman, M., Dolev, S., and Elovici, Y. (2008). Unknown malware detection using opcode representation. In *Proceedings of the IEEE International Conference on Intelligence and Security Informatics*, pages 204–215. Springer.
- Mourtaji, Y., Bouhorma, M., and Alghazzawi, D. (2019). Intelligent framework for malware detection with convolutional neural network. In *Proceedings of the 2nd International Conference on Networking, Information Systems & Security*, pages 1–6.
- Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. (2011). Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*, page 4. ACM.
- Rai, A. (2020). Explainable ai: From black box to glass box. *Journal of the Academy of Marketing Science*, 48(1):137–141.
- Royal, P., Halpin, M., Dagon, D., Edmonds, R., and Lee, W. (2006). Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pages 289–300. IEEE.
- Saxe, J. and Berlin, K. (2015). Deep neural network based malware detection using two dimensional binary program features. In *Proceedings of the 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 11–20. IEEE.
- Schultz, M. G., Eskin, E., Zadok, F., and Stolfo, S. J. (2001). Data mining methods for detection of new malicious executables. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 38–49. IEEE.
- Sebastián, M., Rivera, R., Kotzias, P., and Caballero, J. (2016). Avclass: A tool for massive malware labeling. In *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 230–253. Springer.
- Ye, Y., Li, T., Adjeroh, D., and Iyengar, S. S. (2017). A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)*, 50(3):41.