A Novel Neural Network-based Malware Severity Classification System

Miles Q. Li[1] [0000−0001−7091−3268] and Benjamin C. M. Fung[2] [0000−0001−8423−2906]

[1] School of Computer Science, McGill University, Montreal, Canada
miles.qi.li@mail.mcgill.ca
[2] School of Information Studies, McGill University, Montreal, Canada
ben.fung@mcgill.ca

Abstract. Malware has been an increasing threat to computer users. Different pieces of malware have different damage potential depending on their objectives and functionalities. In the literature, there are many studies that focus on automatically identifying malware with their families. However, there is a lack of focus on automatically identifying the severity level of malware samples. In this paper, we propose a dedicated neural network-based malware severity classification method. It is developed based on the clustering analysis of malware functions. Experimental results show that the proposed method outperforms previously proposed machine learning methods for malware classification on the severity classification problem.

Keywords: cybersecurity · malware severity classification · neural networks

1 Introduction

Malware programs are becoming more sophisticated and diverse with time [16,1]. They are developed for different purposes. Some could harm only individual computers and their users, and the damage can be recovered. Some could cause permanent loss to large groups of computers and their users. Thus, the severity of malware programs can vary. The resources of malware defenders allocated to deal with different malware programs should depend on their severity to minimize the potential losses they can cause. To this end, it is crucial to have an AI-based severity classification system that helps malware analysts recognize the severity level of a malware program in a timely manner.

Signature-based methods are the most commonly used kind of malware classification method in commercial antivirus products. If an executable contains a signature that is labelled with a certain class of malware, it belongs to that class. The signatures are crafted by malware analysts through manual analysis of their collected malware samples. The problem with this type of method is that it is limited in recognizing significant variants of existing malware samples or new malware samples since malware authors could avoid the signature while still keeping its functionalities [42,13,1]. Therefore, machine learning-based malware classification methods are proposed to identify significant variants of known malware or new malware samples based on the patterns that are recognized from known malware samples [36,23,27,35,26].

As a classification problem, malware severity classification is more challenging than malware family classification, while the latter is more intensively studied. One reason that malware family classification is less challenging is that in each malware family, the samples have the same purposes and behaviors, so they are programmed similarly to each other [9,25]. The similarity makes it easier to recognize a new sample of that malware family based on the knowledge of known samples of that family. However, the malware programs at each severity level could present different behaviors and functionalities. Thus, there are many different and independent patterns that can indicate the severity levels of malware programs. This increases the complexity of the malware severity classification problem. The second challenge with malware severity classification is that severity classification is not a normal classification problem in which the relations between all classes are balanced. In severity classification, a higher severity level dominates a lower severity level. In other words, if a program has behaviors at different severity levels, it should be classified as the level of the most severe behaviors. A third challenge is that the malware family classification can be done by analyzing the functional similarity between an unknown sample and a malware family, but severity ranking cannot. When a malware program contains more than the average number of behaviors of programs at a certain level, it should be classified to a higher level, or when most behaviors of a malware program are at a low severity level and only a few behaviors at a high severity level, it belongs to the higher level. For example, when a malware program has multiple Trojan-related functions, it should be classified to a higher severity level than a program that contains only one [21].

The contribution of this paper is a novel and dedicated neural network-based malware severity classification model. It is a neural network extension of the malware family classification model proposed by Li et al. [25], which is based on the similarity analysis of functions of malware samples. Since it is developed based on functional similarity, it will fail the severity classification problems for the reasons we mentioned in the previous paragraph. As artificial neural networks are good at comprehensively capturing the correlation between inputs and outputs rather than just similarity accumulation, we introduce artificial neural networks into the framework proposed by Li et al. [25] for the malware severity classification problem.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 formally defines the malware severity classification problem. Section 4 describes our proposed method. Section 5 presents the evaluation of the proposed method. Section 6 discusses the limitations and our future work. Section 7 concludes the paper.

2 Related Work

2.1 Malware Classification

Most existing malware classification studies focus on malware detection or family classification. The former aims to differentiate malware and benign software, while the latter aims to identify the malware family.

Malware classification methods can be categorized as static, dynamic, and hybrid methods [1]. Static methods examine the static content of an executable, and dynamic

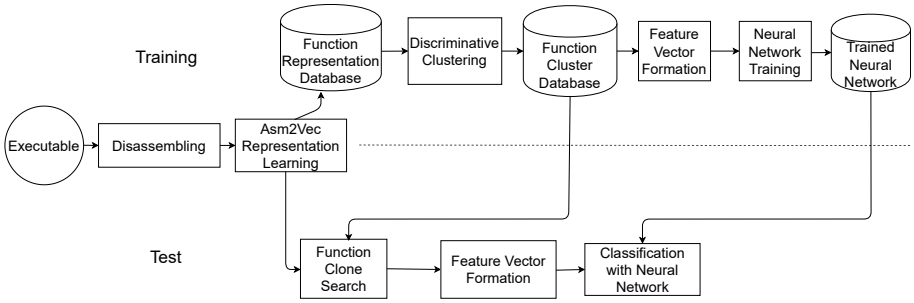

Fig. 1: Workflow of our severity classification model.

methods run it to analyze its behaviors. Common features used by static methods are sequences of bytes [36,23,4,35,32,17], sequences of assembly code [27,11,3,4,34,17,26], numerical PE header fields [4,6,35,26], PE imports/API calls [36,6,35,29,17,26], printable strings [36,20,35,26], and malware images [28,41,40]; those used by dynamic methods are memory images [24,10,19], executed instructions [33,11,3,4], invoked system calls or behaviors [7,15,4,10,20,34,19,2]. Hybrid methods use both static and dynamic features for malware classification [4,20,34,12].

2.2 Severity Ranking

The severity level of malware can be defined based on different criteria. In this subsection, we introduce some severity ranking theories.

Malware Rating System. Bagnall and French [5] suggest three criteria to define the severity of a malware program, namely: (1) its payload potential, (2) its proliferation potential, and (3) its hostility level. The payload means the potential of the code to degrade or damage its target. The proliferation potential means the ability to spread itself across the file system or over the network. The hostility level means how malicious the intent is behind the payload. The payload potential and proliferation potential are classified to 10 ratings and hostility to 5. All three criteria considered, there are 5 severity ratings.

Threat Severity Assessment. Symantec Corporation also suggests three criteria to define severity [38]. They are (1) in-the-wild, (2) damage caused, and (3) rate of distribution. In-the-wild measures the extent to which a virus has already spread among computers. Damage caused measures the amount of damage that a given infection can inflict. The distribution component measures how quickly a program spreads itself. It can be seen that damage caused corresponds to the payload potential and the hostility level in the classification criteria proposed by Bagnall and French [5]. Both in-the-wild and rate of distribution measure the spread of the malware program, with the difference being that the former is about the facts of existing spread and the latter is about

its ability to spread. They correspond to the proliferation potential in the classification criteria proposed by Bagnall and French [5]. Therefore, both of these two severity level definition systems mainly cover two aspects: proliferation and damage.

Kaspersky Lab Threat Level Classification. Kaspersky Lab uses a tree structure to describe their definition of the severity levels of all types of malware programs [21]. Kaspersky Lab does not provide their specific criteria, but it can be seen that the types of malware that are programmed to wildly spread and that may cause tremendous damage are in the upper part of the tree, and the reverse are in the lower part of the tree. This means that the criteria they use are consistent with the first two systems. Kaspersky Lab suggests that the following principles should be taken into consideration to determine the severity: 1) each behavior is assigned its own severity level, and the behaviors that pose less of a threat are outranked by behaviors that pose more of a threat, 2) if a program can be categorized as a number of different behaviors, it should be classified as the most threatening level of those behaviors, 3) if a malicious program has two or more functions with equal severity levels, which could be covered by Trojan Ransom, Trojan ArcBomb, Trojan Clicker, Trojan DDoS, Trojan Downloader, Trojan Dropper, Trojan IM, Trojan Notifier, Trojan Proxy, Trojan SMS, Trojan Spy, Trojan Mailfinder, Trojan GameThief, Trojan PSW or Trojan Banker, then the program will be classified as a Trojan. These principles make sense not only for their severity classification system, but also for the general severity ranking problem. The last principle also makes this classification problem different from normal classification problems in which the relations between all classes are balanced.

3 Problem Definition

Definition 1 (Malware Severity Classification). *Consider a collection of executables E and a collection of labels L that indicate the severity levels of executables in E. Let exe be an unknown executable that $exe \notin E$. The malware severity classification problem is to build a classification model M based on E and L such that M can be used to determine which severity level the executable exe belongs to.* ■

4 Methodology

The workflow of our proposed method is shown in Figure 1. The classification is performed based on the functionality analysis of malware samples. For training the system, we use IDA Pro[3] to disassemble the training samples to get their assembly functions. Then, we apply *Asm2Vec* [14] on the assembly functions to compute their vector representations such that semantically similar assembly functions have large cosine similarities with their vector representations. With the vector representations, we perform a discriminative clustering algorithm to group semantically equivalent assembly functions in a cluster. A feedforward neural network is trained on the vectors representing

[3] https://www.hex-rays.com/products/ida/

whether a sample contains a function that belongs to each cluster as input, and predicts the severity level of the sample. In the test phase, a target sample is disassembled and the vector representations of its assembly functions are computed with the trained *Asm2Vec* model. The vector representations are then used to determine whether the functions belong to a cluster or not. We form a vector representation of the target sample based on whether it has any assembly function that belongs to each cluster. The trained feedforward neural network takes this vector as input to predict the severity level of the target sample.

The disassembling, function representation learning, clustering, and function clone search steps are inherited from the malware family classification system proposed by Li et al. [25]. The feature vector formation and feed-forward neural network classifier are our improvements to that system for the severity classification problem.

4.1 Function Representation Learning

An assembly function consists of one or more blocks of assembly instruction sequences. Assembly functions that achieve the same purpose may appear quite differently when obfuscations or optimizations are applied. Therefore, in its original form, it is hard to directly compare the similarity of assembly functions.

Asm2Vec [14] is a representation learning method for assembly code functions. The vector representations of semantically similar functions have a large cosine similarity so that they can be used to detect clone relations (i.e., similarity larger than a threshold) between different assembly functions. In the training phase, we use the assembly functions of training samples to train *Asm2Vec*, and at the same time, *Asm2Vec* computes the vector representations of the assembly code functions.

The result of this step is the trained *Asm2Vec* and the vector representations of the assembly code functions of the training samples.

4.2 Discriminative Function Clustering

In this step, we put assembly code functions that are semantically equivalent to each other in a cluster. Some clusters are good representatives of certain malware classes since only malware samples from these classes contain assembly code functions that belong to these clusters. They are called discriminative assembly code function clusters. They could be groups of functions related to certain malicious behaviors, such as key logging, proliferation, or corrupting file systems. We identify these clusters with their **discriminative power**. This concept relies on another concept called the popularity of a malware class in a cluster. Let G_i be a cluster, and C_j be a malware class. Let $\|comf(G_i,C_j)\|$ be the percentage of executables in class C_j that has one or more functions in cluster G_i. The **popularity** of malware class C_j in cluster G_i is defined as follows:

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

The discriminative power of cluster G_i is as follows:

$$dp(G_i) = \begin{cases} 0 & \text{if } G_i \text{ contains only 1 function} \\ max_j\{pop(G_i,C_j)\} - min_j\{pop(G_i,C_j)\} & \text{otherwise} \end{cases}$$

In plain words, the discriminative power of a cluster is the difference between the popularity of the class with the maximal popularity and the popularity of the class with the minimal popularity in the cluster. The clusters with high discriminative power characterize the malware classes that have large popularity. In other words, when an executable contains a function that belongs to the cluster, there is a large probability that it belongs to the clusters with large popularity in the cluster as opposed to the rest of the malware classes. Therefore, they can be used to discriminate against the classes with low popularity. On the contrary, in the clusters with low discriminative power, the popularity of all malware classes are similar, thus containing a function of these clusters can not bring much knowledge on which malware classes it is likely to belong to.

To get the discriminative assembly code function clusters from the set of assembly code functions of the training samples, we use a discriminative clustering algorithm [25]. The basic is a Union-Find algorithm to gradually aggregate assembly functions to each cluster and its efficiency is optimized by locality-sensitive hashing (LSH). The LSH function family we use is proposed by Charikar [8]. In the hash function, the only parameter is a random vector $\boldsymbol{r}$, which has the same dimension as the vector representation of an assembly code function. The entries of $\boldsymbol{r}$ are independently drawn from standard Gaussian distribution. The hash value of an assembly code function represented as $\boldsymbol{u}$ is computed as follows:

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

Charikar [8] proved that for two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, the probability that they have the same hash value is as follows:

$$Pr[h_{\boldsymbol{r}}(\boldsymbol{u}) = h_{\boldsymbol{r}}(\boldsymbol{v})] = 1 - \frac{\theta(\boldsymbol{u}, \boldsymbol{v})}{\pi} \tag{2}$$

Therefore, the larger their cosine similarity is, the larger the probability that they have the same hash value. Thus, semantically similar assembly code functions tend to have the same hash values. The way we use LSH to group the assembly code functions can be described as follows:

1. We apply a set of LSH functions on the assembly code functions. The assembly code functions that have the same hash values are put in the same bucket. In each bucket, the assembly functions are potentially equivalent to each other. The number of LSH functions should guarantee that in each bucket, the number of assembly code functions should be smaller than a threshold.
2. We apply the Weighted Quick-Union with Path Compression algorithm [37] on the vector representations of the assembly code functions in each bucket to aggregate them in clusters.
3. We filter the clusters with low discriminative power since they are not informative for discriminating samples of a malware class to other classes.

We refer the readers to the original paper [25] for more details about the discriminative clustering algorithm. We keep the same hyper-parameters as theirs for the algorithm. The idea of processing the training executables for clustering is shown in Figure 2.

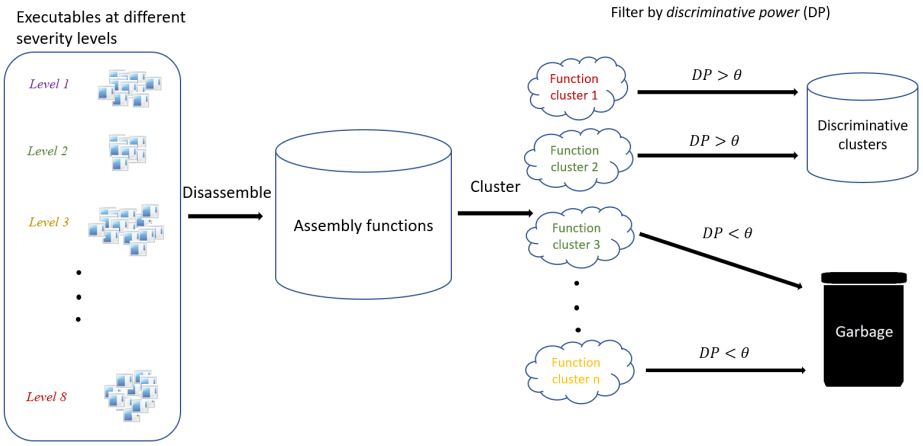

Fig. 2: The procedure to cluster assembly functions.

4.3 Function Clone Search

The classification of a sample is based on its relation to the discriminative assembly code function clusters. The relation of each training sample to the clusters is already known, since the assembly code functions in the clusters are all from training samples. In the test phase, the trained *Asm2Vec* will be applied onto the assembly functions of the test samples to generate the vector representations of the functions. Based on the vector representations, *Asm2Vec* determines whether each assembly function of a test sample is equivalent to an assembly function in a cluster. If it is, the function belongs to that cluster and the test sample is related to the cluster.

4.4 Feature Vector Formation

Let m be the number of discriminative clusters that are formed in the discriminative function clustering step. For each training or test sample, we form a feature vector of dimensions m. Each entry of the vector corresponds to a cluster. The value of an entry is 1 if there is at least one assembly function of the executable that belongs to the cluster (i.e., is equivalent to the functions in the cluster) and 0 otherwise.

4.5 Feed-forward Neural Network Classification

Malware samples in a family are functionally similar to each other. That is the reason that the classification method based on accumulating the functional similarity proposed by [25] could work. However, it would not work for the severity classification problem because similarity does not determine the severity level of a sample. If a malware program contains much more than the average number of functions at a certain level, its severity level is boosted to a higher level. If a malware program contains only a few functions at a higher level and most functions at a lower level, it should still be classified to the higher level [21].

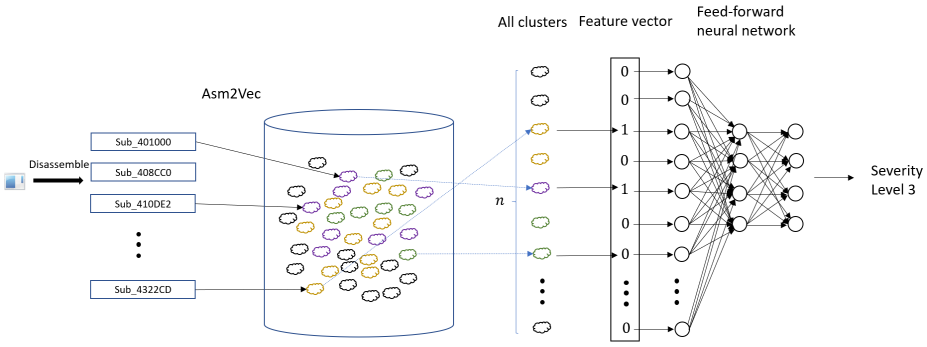

Fig. 3: The procedure to classify an executable.

To solve the aforementioned problem, we replace the functional similarity accumulation-based classification module with an artificial neural network. As is well-known, artificial neural networks are good at pattern recognition for classification. They implicitly learn the patterns that are correlated with each class from the training data. And they can approximate any function to arbitrary accuracy [18]. Therefore, we incorporate a neural network in our proposed malware severity classification model.

The neural network is a feedforward neural network. The input is a feature vector of dimension m formed in the previous step. It is fed to l fully-connected (FC) hidden layers with $Relu$ as the activation function:

$$v_l(x) = FC^l(...FC^1(x)...)$$
$$where\ FC^i(v_{i-1}(x)) = Relu(W_i v_{i-1}(x) + b_i)$$

Then $v_l(x)$ is fed to another FC layer with the output of dimension c, which is the number of classes (i.e., severity levels), and followed by a $softmax$ layer:

$$y(x) = softmax(W_o v_l(x) + b_o) \tag{3}$$

The output $y(x) \in R^c$ is the probability distribution that the query sample is at each severity level. Figure 3 shows the procedure to process an executable and classify it with the feed forward neural network.

In the training phase, we use the feature vectors of training samples and their severity level labels to train the feed forward neural network. We use cross entropy loss as the objective function and Adam [22] as the optimizer with the initial learning rate $1e-4$. The number of hidden layers and the dimensions of the hidden layers are hyper-parameters. We consider 1,2,3 hidden layers and 256,128,64 as the candidate dimensions. We use grid search for tuning hyper-parameters.

In the test phase, we just feed the feature vectors of test samples to the neural network and it computes the probability that the samples belong to each severity.

5 Experiments

In this section, we present the evaluation of our proposed severity classification method. The major evaluation metric is accuracy:

$$accuracy = \frac{number\ of\ correctly\ classified\ samples}{number\ of\ samples\ to\ classify} \tag{4}$$

We also report the precision, recall, and F1 for each severity level (class):

$$precision = \frac{number\ of\ samples\ correctly\ classified\ to\ the\ class}{number\ of\ samples\ classified\ to\ the\ class}$$

$$recall = \frac{number\ of\ samples\ correctly\ classified\ to\ the\ class}{number\ of\ samples\ belonging\ to\ the\ class}$$

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

5.1 Dataset

Table 1: Statistics of the dataset.

Severity	Training		Validation		Test	
	# of exec	# of func	# of exec	# of func	# of exec	# of func
Level 1	169	67,933	56	23,541	56	21,869
Level 2	724	177,361	216	54,849	216	59,178
Level 3	181	10,066	39	2,952	39	3,126
Level 4	181	6,391	53	2,075	53	1,690
Level 5	96	30,443	19	5,650	19	6,719
Level 6	181	2,223	55	344	55	695
Level 7	31	6,178	7	2,712	7	2,909
Level 8	56	5,934	11	1,478	11	2,238
Total	1,619	306,529	456	93,601	456	98,424

Based on the Kaspersky Lab Threat Level Classification tree [21], we create a dataset of 8 severity levels. There are 1619 malware samples in the training set, 456 in the validation set, and 456 in the test set. We use SHA256 checksum to ensure that there is no repetition between those three sets. The statistics of the dataset is given in Table 1. The types of malware included in our dataset at each severity level are shown in Table 2.

Table 2: Types of malware included in each severity level.

Severity	Malware types
Level 1	Hoax,HackTool
Level 2	Trojan-Banker,Trojan-Downloader,Trojan-PSW,Trojan-Ransom,Trojan-Spy
Level 3	Trojan
Level 4	Backdoor
Level 5	Virus
Level 6	Worm
Level 7	Email-Worm
Level 8	Net-Worm

Table 3: Accuracy of different methods on the test set.

Method	Accuracy
Our method	**91.9%**
Mosk2008OpBi	82.2%
Bald2013Meta	90.4%
Saxe2015Deep	87.5%
Mour2019CNN	27.0%
Li2021Func	73.2%

5.2 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. propose to use TF or TF-IDF of opcode bi-grams as features and use document frequency (DF), information gain ratio, or Fisher score as the criterion for feature selection [27]. 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. propose to extract multiple raw features from PE headers and use information gain and calling frequencies for feature selection and PCA for dimension reduction [6]. They apply SVM, J48, and Naïve Bayes as their malware classification models.
- **Saxe2015Deep**: Saxe and Berlin 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 [35].
- **Mour2019CNN**: Mourtaji et al. convert malware binaries to grayscale images and apply a convolutional neural network on malware images for malware classification [28]. Their CNN network has two convolutional layers followed by a fully-connected layer.
- **Li2021Func**: Li et al. 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 [25]. We directly replace malware families with severity levels as the class labels to apply their method to this severity classification problem.

5.3 Experiment Settings

The experiments are conducted on a server with two Xeon E5-2697 CPUs, 384 GB of memory, and four Nvidia Titan XP graphics cards. The operating system is Windows Server 2016.

Our proposed severity classification system and **Li2021Func** are developed with Java 11, and the feedforward neural network is developed with Deeplearning4j [39]. Other baseline methods are implemented with Python 3.7.9. The traditional machine learning models are implemented with scikit-learn 0.23.2 [31], and neural networks are implemented with PyTorch 1.6.0 [30].

5.4 Results

The classification accuracy of different methods is shown in Table 3. Our proposed model achieves the best classification accuracy among all methods. **Bald2013Meta** achieves the second best accuracy, which means the features extracted from PE headers are also informative. However, PE headers only provide peripheral information of an executable, thus, it would not provide as much insight and interpretability as our method since our method is based on the analysis of the malware functionality.

Even though **Li2021Func** is also based on the functionality analysis of malware, it achieves inferior accuracy because of the way it computes the class that a sample belongs to. The severity level of a malware sample is determined by the level of its most threatening behavior, and a greater than average number of behaviors existing in one executable boosts its severity level. However, **Li2021Func** would classify an executable to the level of most behaviors because it is correlated to the most number of clusters at that level. This leads to incorrect classification results.

The precision, recall, and F1 of our model on each severity level is shown in Table 4. Our model performs well for most severity levels except severity level 7. The inferior F1 on level 7 is because we have fewer training samples at severity level 7.

5.5 Classification Result Interpretation

Figure 4 shows an example of the interpretation module of our model. On the left, it lists the assembly functions of a query executable and the function ”sub_408CF3” is selected. On the right, it shows the assembly functions in a cluster that are semantically equivalent to ”sub_408CF3”. They are all from the same cluster ”level_1_Cluster95”, which is a cluster of severity level 1.

Table 4: Experiment results on each severity level.

Severity Level	Precision	Recall	F1-score
Level 1	1.00	0.79	0.88
Level 2	0.87	1.00	0.93
Level 3	0.97	0.90	0.93
Level 4	1.00	0.81	0.90
Level 5	0.83	0.79	0.81
Level 6	1.00	0.95	0.97
Level 7	0.73	0.69	0.71
Level 8	1.00	0.82	0.90
Weighted avg	0.93	0.92	0.92

Fig. 4: An example of interpretation for classification results.

6 Discussions

As is shown in the previous section, our malware severity classification model can explain its classification results by pointing out which functions of the query executable and which function clusters contribute to the classification result. This is the interpretability inherited from the method proposed by Li et al. [25]. However, the neural network module is not directly interpretable. This is a limitation since different functions are not equally important to determining its severity level and it cannot explain how much each assembly function contributes to the classification result. One direction of our future work is to improve the interpretability of the severity classification system so that it can quantify the importance of each assembly function related to its severity.

7 Conclusion

In this paper, to classify the severity levels of malware programs, we propose a neural network-based model that is applied on assembly code function clusters. The method has the same interpretability as the method proposed by Li et al. [25] to point out which

functions contribute to the classification, and it has a better ability to implicitly learn patterns of functionalities to provide accurate severity level estimation of unknown malware samples. It also outperforms previously proposed methods for malware classification on the severity classification task.

Acknowledgment

This research was funded by NSERC Discovery Grants (RGPIN-2018-03872), Canada Research Chairs Program (950-230623), and the Canadian National Defence Innovation for Defence Excellence and Security (IDEaS W7714-217794/001/SV1). The IDEaS program assists in solving some of Canada's toughest defence and security challenges. The Titan Xp used for this research was donated by the NVIDIA Corporation.

References

1. Abusitta, A., Li, M.Q., Fung, B.C.M.: Malware classification and composition analysis: A survey of recent developments. Journal of Information Security and Applications (JISA) **59**(102828), 1–17 (June 2021)
2. Amer, E., Zelinka, I.: A dynamic windows malware detection and prediction method based on contextual understanding of api call sequence. Computers & Security **92**, 101760 (2020)
3. Anderson, B., Quist, D., Neil, J., Storlie, C., Lane, T.: Graph-based malware detection using dynamic analysis. Journal in Computer Virology **7**(4), 247–258 (2011)
4. Anderson, B., Storlie, C., Lane, T.: Improving malware classification: bridging the static/dynamic gap. In: Proceedings of the 5th ACM workshop on Security and artificial intelligence. pp. 3–14. ACM (2012)
5. Bagnall, R.J., French, G.: The malware rating system (mrs)™ (2001)
6. Baldangombo, U., Jambaljav, N., Horng, S.J.: A static malware detection system using data mining methods. arXiv preprint arXiv:1308.2831 (2013)
7. Bayer, U., Moser, A., Kruegel, C., Kirda, E.: Dynamic analysis of malicious code. Journal in Computer Virology **2**(1), 67–77 (2006)
8. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: Proceedings of the 34th annual ACM Symposium on Theory of Computing. pp. 380–388 (2002)
9. Chen, J., Alalfi, M.H., Dean, T.R., Zou, Y.: Detecting android malware using clone detection. Journal of Computer Science and Technology **30**(5), 942–956 (2015)
10. Dahl, G.E., Stokes, J.W., Deng, L., Yu, D.: Large-scale malware classification using random projections and neural networks. In: Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. pp. 3422–3426. IEEE (2013)
11. Dai, J., Guha, R.K., Lee, J.: Efficient virus detection using dynamic instruction sequences. JCP **4**(5), 405–414 (2009)
12. Damodaran, A., Di Troia, F., Visaggio, C.A., Austin, T.H., Stamp, M.: A comparison of static, dynamic, and hybrid analysis for malware detection. Journal of Computer Virology and Hacking Techniques **13**(1), 1–12 (2017)
13. Demontis, A., Melis, M., Biggio, B., Maiorca, D., Arp, D., Rieck, K., Corona, I., Giacinto, G., Roli, F.: Yes, machine learning can be more secure! a case study on android malware detection. IEEE Transactions on Dependable and Secure Computing (2017)
14. Ding, S.H.H., Fung, B.C.M., Charland, P.: Asm2Vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In: Proceedings of the 40th International Symposium on Security and Privacy (S&P). pp. 38–55. IEEE Computer Society (May 2019)

15. Fredrikson, M., Jha, S., Christodorescu, M., Sailer, R., Yan, X.: Synthesizing near-optimal malware specifications from suspicious behaviors. In: Security and Privacy (SP), 2010 IEEE Symposium on. pp. 45–60. IEEE (2010)
16. Gandotra, E., Bansal, D., Sofat, S.: Malware analysis and classification: A survey. Journal of Information Security **2014** (2014)
17. Gibert, D., Mateu, C., Planes, J.: Hydra: A multimodal deep learning framework for malware classification. Computers & Security p. 101873 (2020)
18. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural networks **2**(5), 359–366 (1989)
19. Huang, W., Stokes, J.W.: 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. pp. 399–418. Springer (2016)
20. Islam, R., Tian, R., Batten, L.M., Versteeg, S.: Classification of malware based on integrated static and dynamic features. Journal of Network and Computer Applications **36**(2), 646–656 (2013)
21. Kaspersky, L.: Rules for classifying (2020), https://encyclopedia.kaspersky.com/knowledge/rules-for-classifying/
22. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
23. Kolter, J.Z., Maloof, M.A.: Learning to detect malicious executables in the wild. In: Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD). pp. 470–478. ACM (2004)
24. Kruegel, C., Kirda, E., Mutz, D., Robertson, W., Vigna, G.: Polymorphic worm detection using structural information of executables. In: International Workshop on Recent Advances in Intrusion Detection. pp. 207–226. Springer (2005)
25. Li, M.Q., Fung, B.C.M., Charland, P., Ding, S.H.H.: A novel and dedicated machine learning model for malware classification. In: Proceedings of the 16th International Conference on Software Technologies. pp. 617–628 (2021)
26. Li, M.Q., Fung, B.C., Charland, P., Ding, S.H.: I-mad: Interpretable malware detector using galaxy transformer. Computers & Security p. 102371 (2021)
27. Moskovitch, R., Feher, C., Tzachar, N., Berger, E., Gitelman, M., Dolev, S., Elovici, Y.: Unknown malcode detection using opcode representation. In: Proceedings of the IEEE International Conference on Intelligence and Security Informatics, pp. 204–215. Springer (2008)
28. Mourtaji, Y., Bouhorma, M., Alghazzawi, D.: Intelligent framework for malware detection with convolutional neural network. In: Proceedings of the 2nd International Conference on Networking, Information Systems & Security. p. 7. ACM (2019)
29. Onwuzurike, L., Mariconti, E., Andriotis, P., Cristofaro, E.D., Ross, G., Stringhini, G.: Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). ACM Transactions on Privacy and Security (TOPS) **22**(2), 1–34 (2019)
30. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. Neural Information Processing Systems NIPS 2017 Autodiff Workshop (2017)
31. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
32. Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.: Malware detection by eating a whole exe. arXiv preprint arXiv:1710.09435 (2017)
33. Royal, P., Halpin, M., Dagon, D., Edmonds, R., Lee, W.: Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In: Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC'06). pp. 289–300. IEEE (2006)

34. Santos, I., Devesa, J., Brezo, F., Nieves, J., Bringas, P.G.: Opem: A static-dynamic approach for machine-learning-based malware detection. In: Proceedings of the International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions. pp. 271–280. Springer (2013)
35. Saxe, J., Berlin, K.: 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). pp. 11–20. IEEE (2015)
36. Schultz, M.G., Eskin, E., Zadok, F., Stolfo, S.J.: Data mining methods for detection of new malicious executables. In: Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on. pp. 38–49. IEEE (2001)
37. Sedgewick, R., Wayne, K.: Algorithms. Addison-Wesley Professional (2011)
38. symantec: Severity assessment: Threats, events, vulnerabilities, risks (2006)
39. Team, E.D.D.: DL4J: Deep Learning for Java (2016), https://github.com/eclipse/deeplearning4j
40. Vasan, D., Alazab, M., Wassan, S., Safaei, B., Zheng, Q.: Image-based malware classification using ensemble of cnn architectures (imcec). Computers & Security p. 101748 (2020)
41. Verma, V., Muttoo, S.K., Singh, V.: Multiclass malware classification via first-and second-order texture statistics. Computers & Security **97**, 101895 (2020)
42. Ye, Y., Li, T., Adjeroh, D., Iyengar, S.S.: A survey on malware detection using data mining techniques. ACM Computing Surveys (CSUR) **50**(3), 41 (2017)