Harnessing Open World Machine Learning for Advanced Malware Detection: A Survey

Abhishek Patil[1], Dipmala Salunke[2], Rajvardhan Jadhav[3], Abhishek Shinde[4], Jayesh Shinde[5], and Nilesh Uke[6]

[12345] Rajarshi Shahu College of Engineering, Pune , India
[6] Trinity Academy of Engineering, Pune, India
abhishekpatil56381@gmail.com

Abstract. The ever-changing threat landscape of malware keeps it at the forefront of cybersecurity research. As attackers develop new strains and techniques, researchers continuously work to improve detection methods. This necessitates robust malware classification systems that can not only differentiate between known malware types but also identify entirely new ones. Traditional machine learning approaches, such as static and dynamic analysis, signature-based detection, and classifiers like Support Vector Machines (SVMs), k-Nearest Neighbors (kNNs), and Artificial Neural Networks (ANNs), have been widely used. These methods rely on closed-world learning, assuming all possible classes are present during training. While this works in many scenarios, it falls short in dynamic and evolving environments. The key limitation of these traditional models is their inability to recognize unseen classes - those not encountered during training. This is where open-world machine learning comes in. Open-world learnings empowers systems to adapt to new information, similar to human learning. By enabling the system to recognize and learn from novel threats encountered during operation, open-world machine learning offers a more robust approach to malware classification in the ever-changing real world.

Keywords: malware programs · traditional techniques · signatures · open environments · novel

1 Introduction

Malicious software, commonly known as malware, has long posed a significant threat to computer security, granting attackers varying degrees of control over systems. Given the continuous evolution of malware, there is a pressing need to refine the study of different classification techniques [3]-[7]. Various methods, including those rooted in machine learning [9]- [11] and deep learning [19]-[21], prove instrumental in the analysis and detection of malware. Referencing applicable funding agencies, if available, would enrich the research.

Classical supervised machine learning operates under the closed-world assumption, where in all classes present in testing must have been encountered during training. While this assumption generally holds, it can be violated in dynamic or open-world environments, rendering such models ineffective in certain scenarios.

Open-world learning transcends any particular machine learning paradigm, characterized by the ability to learn from unseen instances not encountered during training while performing specified tasks[1]. Traditional machine learning encompasses five primary tasks: classification, regression, association, clustering, and control, employing various ML techniques. Supervised learning utilizes labeled data for both training and testing, while unsupervised learning operates solely on unlabeled data. Semi-supervised learning harnesses both labeled and unlabeled data. Reinforcement learning, suitable for classification and control tasks, relies on a combination of seen and unseen data for training and testing. Deep learning is adept at both classification and clustering, utilizing seen data for both training and testing. Unlike traditional machine learning, which lacks mechanisms for retaining knowledge across tasks, open-world ma- chine learning incorporates rejection capabilities for unseen instances, testing against seen, seen-unseen, and unseen data. Though often associated with semi-supervised or transfer learning, open-world machine learning distinguishes itself by its ability to classify seen data while rejecting unseen instances. Transfer learning, in contrast, involves knowledge transfer and fine-tuning to classify new data, operating under a closed-world assumption during testing.

2 Literature Review

Table1 highlights the diverse methodologies explored in recent literature, ranging from static and dynamic analysis to advanced machine learning and visualization techniques. Static analysis methods, such as signature-based detection, offer straightforward approaches, whereas dynamic analysis involves testing malware in controlled virtual environments. The integration of image-based classification and deep learning, including the use of convolutional neural networks (CNN) and autoencoders, represents a growing trend in leveraging sophisticated models for malware detection. Additionally, hybrid approaches that combine traditional machine learning with deep learning models, as well as innovative visualization techniques, provide a comprehensive overview of the current state-of-the-art in malware classification research.

As per the survey, various techniques including static, dynamic and signature based analysis are used for distinguish ing different malwares programs. Classification approaches such as Support vector machines(SVM), k-Nearest Neighbors (kNN), and Artificial Neural Networks (ANNs), have been studied. A Convolutional Neural Network (CNN) approach is presented to classify malware patterns when visualized as images. Quick analysis after visualizing through treemaps, thread graphs is studied for detecting and classifying software maliciousness. The prime intend of this study is to improve malware detection using multi-level approach that combines traditional machine learning with deep learning techniques for faster and more accurate analysis.

Table 1. Literatures on different techniques for malware classifications

Paper No.	**Domain**	**Techniques**
3-5	Static Analysis	Signature-based
6, 7	Dynamic Analysis	Testing in virtual environment
8-11	Image-based Classification	Feature Engineering
8	Visualization	Electronics Visualization Technique
9	Deep Learning	Autoencoders
10	Hybrid	Traditional ML + Autoencoder
11	Deep Learning	Convolutional Neural Networks (CNN)
13	Visualization	Self-Organizing Maps
14	Dynamic Analysis	Program execution monitoring
15	Visualization	Software analysis results
16	Visualization	Treemaps, Thread graphs
17	Visualization	Multi-dimensional mapping
19	Deep Learning	Convolutional Neural Networks (CNN)
20	Deep Learning	CNN (compiled files), LSTM (assembly files)
21	Deep Learning	Deep learning models
22	Anomaly Detection	One-class classification with privileged information
23	Classification	Gene sequence method

3 Technologies Used

The breakdown of the technologies represented in the figure 1 is given here, though the specific percentages dependent on total papers referenced during the study.

Deep Learning (25%): This is a powerful machine learning approach that uses artificial neural networks to learn complex patterns from data. Deep learning models can be very effective at detecting malware, especially novel or zero-day malware that hasn't been seen before.
Static Analysis (20%): This technique involves examining the code of a program to identify suspicious patterns that might indicate malware. Static analysis can be quick and efficient, but it can't detect malware that doesn't reveal itself in the code.
Dynamic Analysis (15%): This technique involves running a program in a safe sandbox environment and observing its behavior. Dynamic analysis can be more effective at detecting malware that hides its malicious behavior in the code, but it can be slower and more resource-intensive than static analysis.
Image-Based Classification (10%): This technique involves converting malware samples into images and using machine learning algorithms to classify them as malicious or benign. This approach can be effective for detecting malware that spreads through email attachments or other visual media.
Visualization (15%): This technique involves using data visualization techniques to help analysts understand the characteristics of malware and identify new trends. Visualization can be a valuable tool for improving the effectiveness of other malware detection techniques.
Other (5%): This category includes other techniques that are used less frequently for malware detection and classification, such as gene sequence method, treemaps and thread graphs etc.

4 Methodology

In this paper, we propose a model that has ability to work on classification of malwares that has seen during the training and when an unseen instance of malware is detected, it should reject it. For this, assembly and compiled files provided in the dataset are leveraged. After visualizing these malware programs as images, the open classification network (traditional classification with rejection capability), model is used to classify them or reject it. The auto-encoders are used to learn representations from the rejected instances of malware programs. At final stage, unseen classes are discovered using the clustering technique.

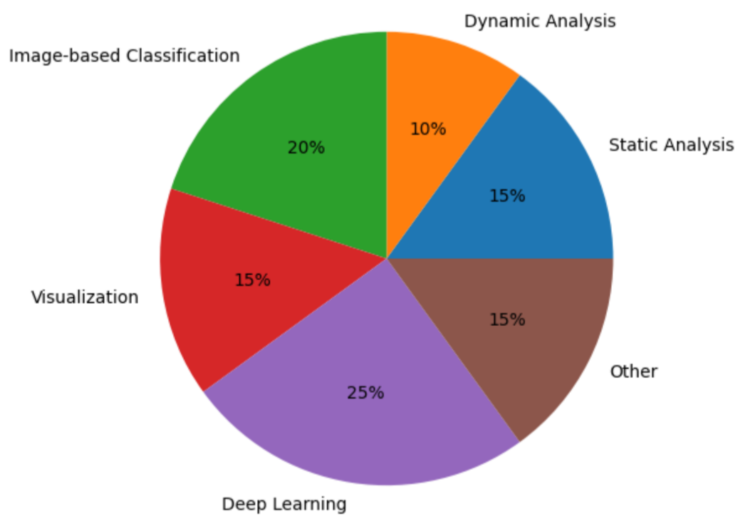

Fig. 1. Representation of different domains in malware detection and classification

4.1 Data Gathering and Preprocessing

For this task, we are leveraging on Microsoft Malware Classification Challenge (BIG 2015) dataset which is available on Kaggle [18]. Figure 2 describes the distribution of various categories of malware in the dataset.

4.2 Feature Extraction

For each malware program, the dataset contains an assembly file (Human-readable instructions for processor converted by assembler) and compiled file (Machine code directly executable by processor). The preprocessing of both file is done seperately.

Compiled file: The compiled file undergoes a transfor mation into an image, wherein each 2-digit hexadecimal code is converted into its corresponding decimal number, resulting in a gray-scale image with a wide range of shades. Visualization patterns for each category in the dataset are depicted in Figure 3.
Assembly file: Assembly files are stripped down to adjust their opcodes (instructions for the proccessor). The order these opcodes appear in is maintained maintained reflecting the program flow. The identify 483 unique opcodes and analyze how often each appears in different malware categories . opcode frequency is visualized as a word cloud for each malware category ,with larger conts indicating more frequent opcodes. Figure 4 represents word cloud for these files.

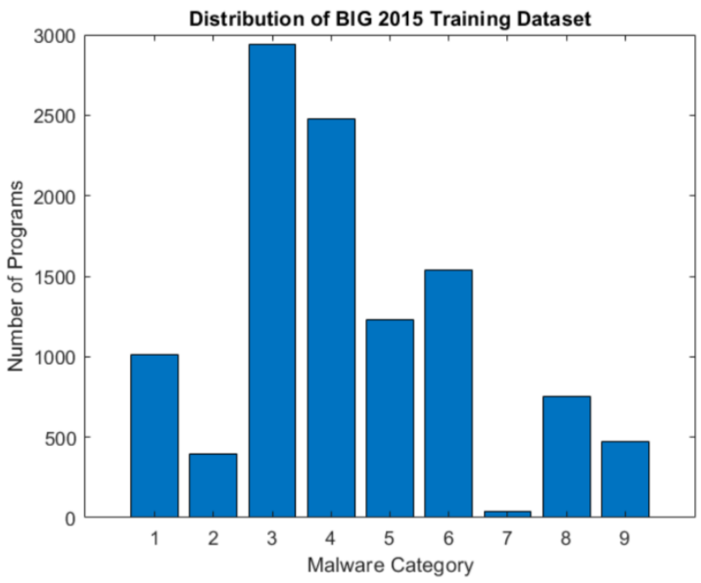

Fig. 2. Visualizing malware Categories in BIG 2015 Dataset

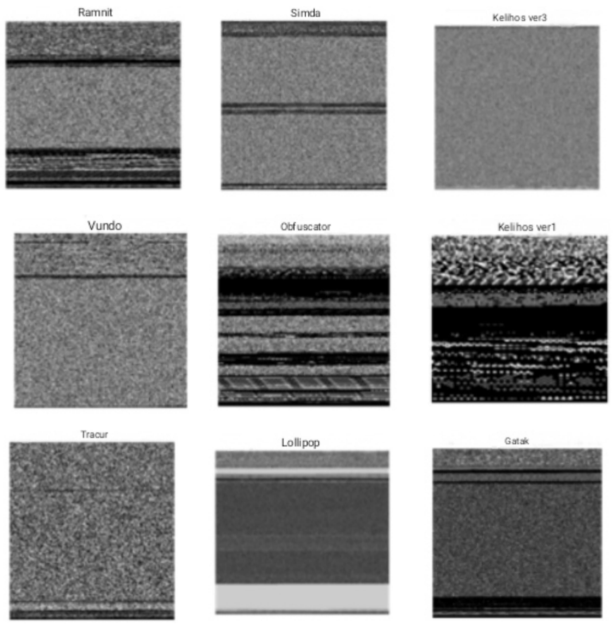

Fig. 3. Visualizations of different malware programs

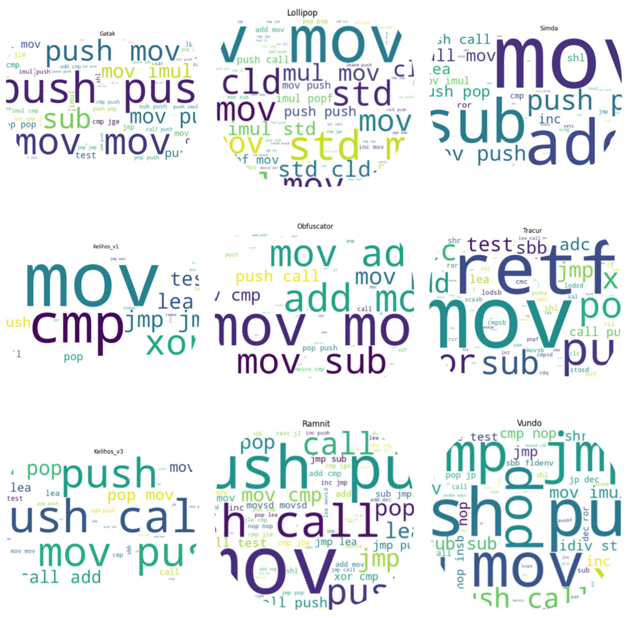

Fig. 4. Word cloud of malware programs

4.3 Model

As seen in [24], considering the annotated training dataset,

$$D = \{(x_1, y_1), (x_i, y_i), \ldots, (x_n, y_n)\}$$

comprising examples from m identified categories, where xi represents the i-th instance with its corresponding label yi belonging to S, denoting the set of class labels, our objectives are to, (a) construct a predictive model f(x) capable of categorizing each new instance x into one of the m known classes in S or designate it as unfamiliar, (b) establish a binary classifier g(xp, xq) to ascertain whether two instances (xp and xq, irrespective of their category) belong to the same class or not, and (c) for all unclassified instances, ascertain the number of latent classes they pertain to based on g(xp, xq).

1. Open Set Classification Network (OCN): This network per forms open classification, assigning test examples to known classes or rejecting them if they belong to unseen classes.
2. Pairwise Classification Network (PCN): This network takes two input examples and predicts whether they belong to the same class or not, irrespective of their origin (known or unknown class).
3. Autoencoder: This component learns meaningful representa tions from unlabeled data, which can be helpful for both OCN and PCN.

4. Hierarchical Clustering: Based on the pairwise class relation ships learned by PCN, this step clusters the rejected examples from OCN. This helps estimate the no. of hidden classes potentially embedded within the rejected data points.

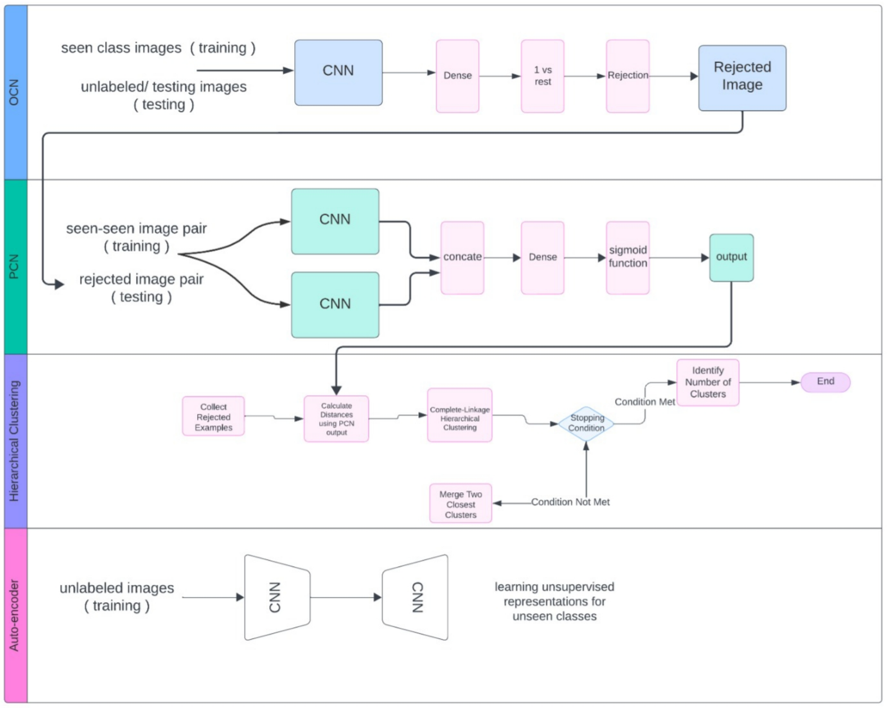
OCN
seen class images (training)
unlabeled/ testing images (testing)
CNN
Dense
1 vs rest
Rejection
Rejected Image
PCN
seen-seen image pair (training)
rejected image pair (testing)
CNN
CNN
concate
Dense
sigmoid function
output
Hierarchical Clustering
Collect Rejected Examples
Calculate Distances using PCN output
Complete-Linkage Hierarchical Clustering
Stopping Condition
Condition Met
Identify Number of Clusters
End
Condition Not Met
Merge Two Closest Clusters
Auto-encoder
unlabeled images (training)
CNN
CNN
learning unsupervised representations for unseen classes

Fig. 5. proposed architecture

5 Workflow

A. Open Set Classification Network (OCN): OCN firstly, leverages a shared Convolutional Neural Network (CNN) to extract features from the data. This extracted feature representation is then sent into fully-connected layer to further processing. Critically, instead of the typical softmax output layer used in closed-set classification, the OCN uses a 1-vs-rest layer with sigmoid activations. This allows each element in the layer to output a probability between 0 and 1, indicating the likelihood of the input belonging to a specific seen class. The key to identifying unseen classes lies in the thresholding mechanism. OCN utilizes an adaptive threshold based on outlier detection principles within a Gaussian distribution framework. This approach goes beyond a simple fixed threshold (like

0.5) and allows for more robust rejection of examples that don't confidently map to any seen class. The details of this specific threshold calculation can be found in the original research paper.

B. Pairwise Classification Network (PCN): Pairwise Classification Network (PCN): Next, The PCN tack les the challenge of understanding both the similarities within known classes (seen classes) and the differences between them. This knowledge will later be crucial for identifying unseen classes during the clustering phase. The network's architecture revolves around two identical Convolutional Neural Network (CNN) branches that process data independently. Once pro cessed, the outputs from these branches are combined and fed through two fully-connected layers for further analysis. Finally, a single layer with a sigmoid activation function tack les the core task: predicting whether the two input examples, one from each branch, belong to the same class. Training PCN relies on a strategy that leverages pairs of examples. The system feeds the network positive pairs, where both examples belong to the same class (intra-class), allowing it to learn the subtle nuances within a known category. On the other hand, negative pairs consisting of examples from different classes (inter-class) are also presented. This equips PCN with the ability to distinguish between distinct seen classes.

However, directly using all possible pairs for training be comes impractical with a large number of seen classes. To address this, the system employs a sampling strategy that uniformly selects pairs from both intra-class and inter-class categories. Finally, a binary log loss function, monitors the training process, gauging its efficiency based on the predic tions made for all sampled pairs.

C. Autoencoder: Joint Training: In this phase, we employ an auto-encoder to simultaneously acquire unsupervised representations for the unseen instances derived from unlabeled examples. This approach is necessary as relying solely on observed instances might result in overfitting to those instances alone, conse quently yielding poor representations for unseen categories. We integrate the training of OCN, PCN, and the auto-encoder into a multi-task learning procedure, with the cumulative loss of this unified optimization being:

$$\mathcal{L}_{joint} = \mathcal{L}_{OCN} + \mathcal{L}_{PCN} + \mathcal{L}_{ae} :$$

$$\mathrm{L}_{joint} = \sum_{i=1}^{m} \sum_{j=1}^{n} -\delta_{Yi,sj} \log p(Yi = sj) - I\{yi \neq sj\} \log(1 - p(Yi = sj)),$$

$$\mathrm{L}_{PCN} = \sum_{i=1}^{n} \mathbf{1}\{yi = 1\} \log p(y = 1) + \mathbf{1}\{yi = 0\} \log p(y = 0),$$

$$\mathrm{L}_{ae} = \sum_{i=1}^{n} E_i.(1)$$

D. Hierarchical Clustering: Hierarchical clustering offers a dis tinct advantage over k-means clustering by eliminating the need to specify the number of clusters beforehand. The ob jective here is to ascertain the number of clusters within the rejected examples and subsequently characterize these clusters.

The fundamental principle underlying hierarchical clustering involves iteratively merging two clusters until a predefined stopping criterion, based on a distance function d, is met. In this context, the output of PCN serves as a viable distance function, having been trained to distinguish between same and different class instances. We opt for complete-linkage hierarchical clustering, wherein the distance between two clusters, C1 and C2, is determined by the maximum distance between any pair of examples belonging to each cluster. This distance function is formally expressed as:

$$d(C_1, C_2) = \max_{x_p \in C_1,\, x_q \in C_2} g(x_p, x_q) \tag{2}$$

Given our objective of determining the number of clus ters corresponding to the hidden, unseen classes, a distance threshold serves as a crucial stopping criterion to prevent further merging of clusters. While sigmoid function used in PCN assumes a threshold value 0.5, this may not be optimal for unseen classes. To address this, we reserve a subset of classes, denoted as validation classes V, from the seen classes to facilitate a more appropriate threshold selection for hierarchical clustering. Once hierarchical clustering converges to the true number of clusters represented by V, we utilize the largest distance between any two clusters, C i and C j , as the threshold.

$$\theta_i \leq |V|, j \leq |V|, i \neq j \max_{i,j} d(C_i, C_j) \tag{3}$$

6 Observation

This paper explores various classification methods to dif ferentiate between malware programs based on available data types. Neural network technologies, such as CNN, demonstrate superior accuracy in feature extraction compared to static, dynamic, and visualization methods. Given the continual evo lution of malware, anti-malware companies must continuously evolve to outpace malicious developers. Therefore, there is a need for self-learning systems capable of updating themselves autonomously as they perform their designated tasks, without explicit training.

7 Challenges

The dataset (BIG 2015) having 9 categories of malware programs, is comparatively older. The performance of the model might vary with the new latest data. Also the malware categories might be higher compared to the dataset used in this paper9. The dataset (BIG 2015) having 9 categories of malware programs, is comparatively older. The performance of the model might vary with the new latest data. Also the malware categories might be higher compared to the dataset used in this paper.

8 Conclusion

The survey analyzed different techniques for classification and detection of different malware programs. This paper introduces a combined model for malware classification. It can not only categorize images into known classes (seen classes) but also predict if two images belong to the same class, regardless of being seen or unseen. This unique ability allows the model to leverage knowledge about class similarity from known data to unseen data. The model then utilizes a hierar chical clustering technique to identify potential hidden classes (groups) within the unclassified images (rejected examples). The study shows that the model is highly promising.

References

1. JITENDRA PARMAR and SATYENDRA S. CHOUHAN, Malaviya National Institute of Technology Jaipur, INDIA. Open-world Machine Learning: Applications, Challenges, and Opportunities
2. Barath Narayanan Narayanan, Venkata Salini Priyamvada Davuluru. Ensemble Malware Classification System using Deep Neural Networks
3. Tian, R.; Batten, L.M.; Versteeg, S.C. Function length as a tool for malware classification. In Proceedings of the 2008 3rd International Con ference on Malicious and Unwanted Software (MALWARE), Fairfax, VI, USA,
4. Karim, M.E.; Walenstein, A.; Lakhotia, A.; Parida, L. Malware phy logeny generation using permutations of code. J. Comput. Virol.
5. Kolter, J.Z.; Maloof, M.A. Learning to detect malicious executables in the wild. In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (ACM), Seattle, WA, USA,
6. Park, Y.; Reeves, D.; Mulukutla, V.; Sundaravel, B. Fast malware classification by automated behavioral graph matching. In Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research (ACM), Oak Ridge, TN, USA, 21–23 April 2010 using structured control flow. In Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing, 18 January 2010; Australian Computer Society.
7. Narayanan, B.N.; Djaneye-Boundjou, O.; Kebede, T.M. Performance analysis of machine learning and pattern recognition algorithms for malware classification. In Proceedings of the 2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS), Dayton, OH, USA, 25–29 July 2016; pp. 338–342
8. Kebede, T.M.; Djaneye-Boundjou, O.; Narayanan, B.N.; Performance analysis of machine learning and pattern recognition algorithms for Malware classification; 10.1109/NAECON.2016.7856826 Performance a
9. Cesare, S.; Yang, X. Classification of malwareescu, A.; Kapp, D. Classification of malware programs using autoencoders based deep learning architecture and its application to the microsoft malware
10. Messay-Kebede, T.; Narayanan, B.N.; Djaneye-Boundjou, O. Combi nation of Traditional and Deep Learning based Architectures to Over come Class Imbalance and its Application to Malware Classification. In Proceedings of the NAECON 2018-IEEE National Aerospace and Electronics Conference, Dayton, OH, USA, 23–26 July 2018; pp. 73–77.

11. Davuluru, V.S.P.; Narayanan, B.N.; Balster, E.J. Convolutional Neural Networks as Classification Tools and Feature Extractors for Distin guishing Malware Programs. In Proceedings of the 2019 IEEE National Aerospace and Electronics Conference (NAECON), Dayton, OH, USA, 15–19 July 2019; pp. 273–278.
12. Nataraj, L.; Karthikeyan, S.; Jacob, G.; Manjunath, B.S. Malware images: Visualization and automatic classification. In Proceedings of the 8th International Symposium on Visualization for Cyber Security (ACM), Pittsburgh,
13. Yoo, I. Visualizing windows executable viruses using self-organizing maps. In Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, Washington, DC, USA, 29
14. Quist, D.A.; Liebrock, L.M. Visualizing compiled executables for mal ware analysis. In Proceedings of the 2009 6th International Workshop on Visualization for Cyber Security (IEEE), Atlantic City, NJ, USA
15. Goodall, J.R.; Radwan, H.; Halseth, L. Visual analysis of code security. In Proceedings of the Seventh International Symposium on Visualization for Cyber Security (ACM), Ottawa, ON, Canada
16. Trinius, P.; Holz, T.; G¨obel, J.; Freiling, F.C. Visual analysis of malware behavior using treemaps and thread graphs. In Proceedings of the 2009 6th International Workshop on Visualization for Cyber Security, Atlantic City
17. Conti, G.; Bratus, S.; Shubina, A.; Sangster, B.; Ragsdale, R.; Supan, M.; Lichtenberg, A.; Perez-Alemany, R. Automated mapping of large binary objects using primitive fragment type classification. Digit. Investing
18. Kaggle BIG 2015 Dataset. https://www.kaggle.com/c/malware-classification Available online:
19. Ronen, R.; Radu, M.; Feuerstein, C.; Yom-Tov, E.; Ahmadi, M. Mi crosoft malware classification challenge. arXiv 2018, arXiv:1802.10135. Available online: https://arxiv.org/abs/1802.10135
20. Yan, J.; Qi, Y.; Rao, Q. Detecting malware with an ensemble method based on deep neural network. Secur. Commun. Netw. 2018
21. Garcia, F.C.C.; Muga, I.I.; Felix, P. Random forest for mal ware classification. arXiv 2016, arXiv:1609.07770. Available online: https://arxiv.org/abs/1609.07770 .
22. Burnaev, E.; Smolyakov, D. One-class SVM with privileged information and its application to malware detection. In Proceedings of the 2016 IEEE 16th International Conference on Data Mining Workshops
23. Drew, J.; Moore, T.; Hahsler, M. Polymorphic malware detection using sequence classification methods. In Proceedings of the 2016 IEEE Security and Privacy Workshops (SPW), San Jose, CA, USA, 22–26 May 2016
24. Lei Shu, Hu Xu, Bing Liu, unseen class discovery in open-world classification,arXiv:1801.05609v1 [cs.LG] 17 Jan 2018, pp.3-5