Evolutionary Deep-Learning Malware Classifiers

Sabre Didi, Geoff Nitschke
Department of Computer Science
University of Cape Town
Cape Town, South Africa
ddxsab001@myuct.ac.za, gnitschke@cs.uct.ac.za

***Abstract*—Malware attacks remain a critical cyber-security concern, necessitating robust solutions for both individual users and organizations. Deep learning methods have become pervasive tools for malware detection and classification. However, the evolution of malware into sophisticated forms that aim to elude detection poses a formidable challenge to traditional deep-learning methods. Existing techniques for generating adversarial samples often rely on manual feature extraction and white-box models, introducing a gap between the generated samples and real-world scenarios. In response to these challenges, we propose an innovative approach leveraging evolutionary learning for the generation of adversarial samples. Our approach uses a three-step process for malware detection. First, a trained deep-learning malware classifier categorizes samples as benign or malicious. Second, an evolutionary adversarial learning approach trains and generates new malware samples. Third, competitive co-evolution facilitates automated adaptation of malware detection agents that are robust against attacks. We evaluate the efficacy of our approach for adaptive malware detection via benchmark evaluations with an established deep-learning classifier.**

I. Introduction

Recently, the surge in malware threats has prompted significant research into malware detection methods, with a particular focus on dynamic and static analyses [1], [2]. Static analysis entails analyzing features of binary programs without their actual execution, as first proposed by Lo et al. [3]. While less resource-intensive and more secure than dynamic analysis, static analysis may not capture real-time malware behavior accurately. Whereas, dynamic analysis involves extracting features by monitoring program executions, making it an intuitively preferable choice for obtaining accurate data on program behavior [4]. However, this approach comes with practical challenges, requiring a tailored runtime environment like a customized *Virtual Machine* (VM), which can be computationally expensive, especially with a large number of samples. Furthermore, although malware classification research spans multiple platforms, Windows remains a common environment, and most threats revolve around Windows PE malware [5], and as such this study focuses on adaptive classifiers for Windows PE malware.

To address mounting security threats posed by Windows PE malware, extensive research efforts have been dedicated to effective and efficient detection. Traditional signature-based detection [6], which identifies suspicious software by comparing its signature to a known malware database, has been the historical norm. However, the main limitation of such methods is in detecting only previously identified malware, relying heavily on malware databases [5]. In the last decade, inspired by the significant advances in *Deep Learning* (DL) [7] across various real-world applications (such as computer vision, natural language processing, and speech recognition), a range of DL-based malware detection methods have emerged, demonstrating effective detection of various Windows PE malware. The effectiveness of such DL-based methods is based on generalization capacity, predicting new and previously unseen (*zero-day*) instances of malware [8]. Recent studies have exposed the vulnerability of various DL methods to adversarial attacks, specifically through meticulously crafted adversarial examples [5], [9]. These examples involve slight manipulations of legitimate inputs to deliberately confuse specific DL classifiers.

Many adversarial attacks rely on feature-space attacks, utilizing various gradient-based methods to, for example, generate adversarial images [10]. Security researchers and practitioners in academia and industry have proposed numerous attack methods and defense mechanisms over time, addressing the growing concerns of DL method vulnerabilities and bolstering them against increasingly sophisticated adversarial landscapes.

The majority of research in malware classification has traditionally centered around manually crafted perturbations, as exemplified by recent publications introducing concepts such as the *side-effect* feature arising from the inverse-mapping problem [9]. However, this study takes the approach of evolving malware classifiers adapted with evolving adversarial malware samples. This approach aims to test the hypotheses we posit. First, neuro-evolution coupled with *Convolutional Neural Network* (CNN) training achieves classification accuracy comparable to an established DL classifier. Second, CNNs trained and co-evolved with adversarial malware samples result in classifiers with consistently high classification accuracy (robust to adapting malware). The key contribution of this study is the demonstrated efficacy of competitive co-evolution (malware detection versus malware evasion), for the automated adaptation of DL malware classifiers, where such co-evolved classifiers perform comparably to an established DL classifier while yielding consistently high task performance (malware detection) given continually adapting malware samples.

II. Methods and Experiments

Methods [11] inter-leave training with malware samples and evolving DL classifiers (section II-B) to automate adaptation to concurrently evolving adversarial malware samples (perturbations of a malware database, section II-A, II-C).

A. Malware Adversarial Detection Model

Figure 1 illustrates this study's malware versus DL classifier co-evolution model. Specifically, the population of CNN classifiers are initially trained on the malware dataset. CNNs are then adapted via neuro-evolution to synthetic malware dataset perturbations and the CNNs are then re-trained for the malware perturbations. The next cycle of malware perturbations and CNN neuro-evolution [12] to adapt these perturbed malware samples then begins. Thus the malware samples undergo simultaneous evolution while the malware detection process introduces perturbations (applied to 80% of the malware dataset) as per equation 1, after which CNNs are retrained. When the detection accuracy surpasses a defined threshold (Table I), malware sample perturbations are applied again, and the re-training and evolution cycle repeats. Adversarial sample extraction (perturbation) is given in equation 1.

$$x_* = x + \delta_x = x + min||z|| \ s.t. \ F(x+z) \neq (F(x) \qquad (1)$$

Where, sample x is perturbed with δ_x (an added update), making a new sample $F(x+z)$ from the original sample $F(x)$.

B. Evolving Deep Learner (DL) Classifiers

The individual (genotype evolved) in this study is a DL malware classifier, represented as a fully connected feed-forward *Convolutional Neural Network* (CNN), using non-linear activation functions across the following sequence of layers: *convolution*, *max-pooling*, *convolution*, *max-pooling*, *flattening*, *dense*, *drop-out*, and *dense* (layers are described in previous work [13]). In our experiments an input layer of 77 features was connected hidden layers (initialized as follows): an 79x256 convolution layer, 39x256 max-pooling layer, 39x128 convolution layer, 19x128 max-pooling layer, 1x2432 flatten layer, 1x512 dense layer and 1x512 drop-out layer, connected to two outputs (classification: *malware*, *benign*). Table I presents the CNN architecture, hyper-parameters and neuro-evolution parameters. Parameters not described here can be found in related work [11]. To remove irrelevant features and ensure consistency across all input samples (section II-C) we applied principal component analysis to reduce the number of CNN inputs (features) to 77.

We applied NEAT [12] to evolve the hidden layer connectivity and weights for several CNN layers (CNN parameters evolved are given in Table I). A population of 150 CNN classifiers were implemented using the NEAT-Python framework[1], integrated with *Weight Agnostic Neural Networks* (WANN) [14]. WANN search samples a single shared weight at each roll-out to explore neural network topology. CNN fitness is evaluated across multiple roll-outs, ranked according to maximal classification performance (portion of correct malware classifications, Table II). Per generation of NEAT, the fittest 20% of CNNs undergo NEAT variation operations [12], where the child population produced by this fittest 20% replaces the current population. One evolutionary run is 100 generations of this neuro-evolution process, where evolved DL classifier task performance is an average over 20 runs (Table I).

C. Dataset

The population of CNN classifiers was trained using a dataset [11] comprising both malicious and benign program data derived from *Windows Portable Executable* (PE) files. The dataset, curated from Kaggle[2], included 19611 malicious samples sourced from diverse malware repositories such as *VirusShare*. The PE file format, integral to Win32 specifications, was introduced by Microsoft to facilitate program execution on the Windows operating system. Given compatibility with most versions of the Windows OS, PE files have emerged as the predominant conduit for malware propagation [15]. Our dataset, where samples comprised 77 features, encompassed:

- *NumberOfSections:* Section table size (directly succeeds headers) − different in malware and non-malware files.
- *MajorLinkerVersion:* Linker version number.
- *AddressOfEntryPoint:* Optional field header and entry point address. Address is for the image base obtained as the PE file is loaded into memory − starting address for program images and initialization function address for device drivers. The field is null when there is no entry point. No entry point for a Dynamic-link Library (DLL).
- *ImageBase:* Address of the first byte of the image when it is loaded into memory (usually a multiple of 64K).
- *MajorOperatingSystemVersion:* Number used to identify the version of the operating system.
- *MajorImageVersion:* Number used to identify the version of the image. Many benign files have more versions and most malicious files have this feature with a value of zero.
- *CheckSum:* 90% of the time, when the *CheckSum*, *MajorImageVersion*, and *DLLCharacteristics* of a file are equal to zero, the file is found to be malicious.
- *SizeOfImage:* Image size as it is loaded in memory.

D. Experiments

Experiments aimed to validate the following hypotheses:

- Neuro-evolution coupled with CNN training achieves malware classification accuracy comparable to established DL malware classifiers.
- CNNs trained and co-evolved with adversarial malware samples result in DL classifiers with consistently high classification accuracy (robust to adapting malware).

Our malware dataset comprises Windows PE binary files, where the dataset, denoted as D, comprises x classes, each with a size of N instances (section II-C). The training set, T, is a subset of a specific class within D and is employed to

[1]https://pypi.org/project/neat-python/

[2]https://www.kaggle.com/datasets/amauricio/pe-files-malwares

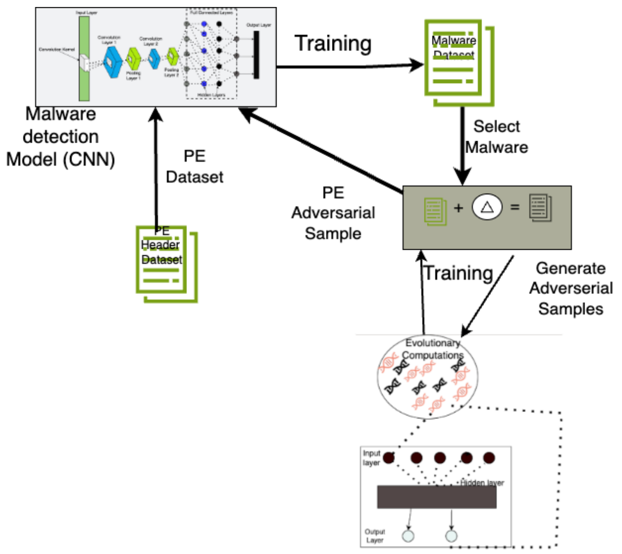


Fig. 1: Malware detection: Deep-learning classifier is trained with continually generated adversarial (malware) examples.

TABLE I: Method and Experiment Parameters.

Evolving DL Classifier (CNN)	
Parameter	**Value**
Input, Output Layer Size	77, 2
Hidden Layers	7
Hidden Layer Size	[2, 79x256]
Weight Values	[0.0, 1.0]
Epochs	100
Batch Size	1000
Activation Functions	Sigmoid, Tanh
Maximum parameters	1346178
Benchmark CNN	
MalConv architecture	Chen et al. [16]
Evolutionary Algorithm (NEAT-WANN)	
Number of Generations	100
Population size	150
Detection accuracy threshold	80%
Stop (Fitness threshold)	0.99
Activation Mutation Rate	0.2
Parent selection/replacement	Fittest 20%, Generational
Experiments	
Training data	Section II-C
Training, validation, testing	80%, 10%, 10% data subsets
Initialization (weights)	Random
Runs	20
Evaluation (fitness) metric	Table II

train the elements belonging to that given class (equation 2). This ensures a training process that enhances CNN capabilities to discern patterns and characteristics in each class [17]. Method and experiment parameters are presented in table I. The classification accuracy (fitness) of trained and evolving CNN classifiers used the metrics detailed in table II.

$$T_i \subseteq N_i \subseteq D_i, i \in \{1, 2, 3..x\} \tag{2}$$

Experiments compared the training and validation *accuracy* of a benchmark CNN classifier (*MalConv* architecture [16]), demonstrated as especially effective for detecting various malware. The comparison was between a trained benchmark CNN classifier and the best evolved CNN (highest validation *accuracy*). All architecture and hyper-parameters of the benchmark CNN are as stated in related work [16]. The comparative task performance metric (classification accuracy) was as an average over 20 runs (section III), computed given 20 random weight initializations and executions of the benchmark CNN and 10 evolutionary runs of NEAT-WANN (section II-B).

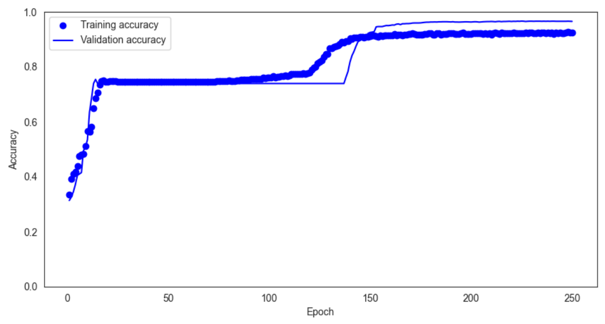
III. Results and Discussion

Results indicate that co-evolving CNN malware classifiers with adversarial malware samples (PE files) yields (on average) faster convergence to high classification accuracy (Figure 2, right) compared the benchmark CNN (Figure 2, left). Our co-evolution approach yields an average (20 runs) validation accuracy of 0.97 after approximately 75 epochs, whereas, the benchmark CNN yields a validation accuracy of 0.90 after approximately 150 epochs. Furthermore, statistical tests, (Mann–Whitney U, $p<0.05$, [18]) indicated a statistically significant difference between average training and validation accuracy's of the benchmark CNN versus the CNN co-evolved with malware samples, where the co-evolution approach achieved higher accuracy overall.

These results support our hypotheses (section II-D) and underscore the benefits of co-evolving classifiers and synthetic malware (approximating perpetually emerging real-world malware). Specifically, these results support the efficacy of co-evolving CNN architectures competitively with malware samples (synthetic variants of a Windows PE file database).

TABLE II: Malware classification metrics used for all CNN classifiers.

Metric	Formula	Description
True Positive (TP)	$tp = \sum_1^n$	Total malware correctly predicted
False Positive (FP)	$fp = \sum_1^n$	Total benign incorrectly predicted
True Negative (TN)	$tn = \sum_1^n$	Total benign correctly predicted
False Negative (FN)	$fn = \sum_1^n$	Total malware incorrectly predicted
Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$	Rate of correct predictions



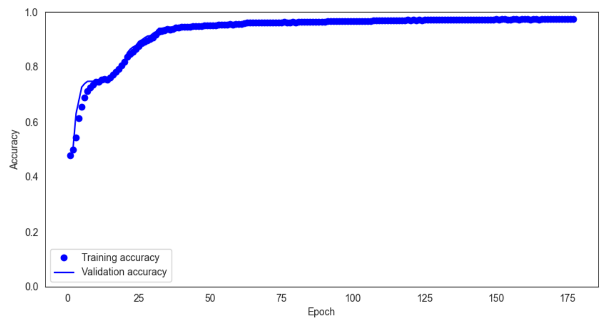


Fig. 2: Left: Average (20 runs) training and validation *accuracy* (table II) of benchmark CNN classifier. Right: Average (20 runs) training and validation *accuracy* of CNN co-evolved with adversarial malware samples.

Results were further supported by comparisons with an established CNN classifier trained on current malware data. That is, the lower average accuracy yielded by the benchmark CNN is a result of training and validation on a fixed dataset (section II-C). In the latter case, datasets must be constantly updated with the latest malware variants, where even re-trained classifiers with careful hyper-parameter tuning still mis-classify new malware [1], [2]. The key contribution is thus demonstration of quick convergence to a high classification accuracy given adapting (co-evolving) malware samples (section II-D). This has significant potential benefits for future automated malware applications that must rapidly adapt to constantly emerging variants of malware across a broad range of applications and operating systems. Given that malware datasets for training classifiers become quickly outdated and the emergence of malware is currently outpacing many malware database updates, co-evolutionary classifiers, specially adapted with co-evolving adversarial malware samples, are a key alternative for malware classification.

With notable exceptions [19], the use of competitive co-evolution to adapt malware detectors versus adversarial malware samples [20] has been little investigated, though such related work supports this study's demonstrated efficacy of using competitive co-evolution as a means to boost the average accuracy of malware classifiers. Also, to the authors' knowledge this study is the first example of applying competitive co-evolution to adapt (via neuro-evolution) CNN classifiers in company with synthetic malware samples (evolved from a given malware dataset of Windows PE files).

IV. Conclusion

This study demonstrated and evaluated the competitive co-evolution of CNN classifiers in a company with synthetic adversarial malware samples. CNN classifiers were evaluated according to malware classification accuracy and malware samples were evaluated according to evasion of correct CNN classification. This study's key contribution was demonstration of co-evolving CNN classifiers and synthetic malware examples as an effective means to achieve consistently high malware classification accuracy for constantly adapting malware samples (co-evolving with classifiers). The classification accuracy of the fittest evolved CNN classifier significantly exceeded that of an established CNN (benchmark) classifier trained on the same malware data, supporting the efficacy of competitive co-evolution as an effective means for continually adapting CNN malware classifiers in response to new emergent malware. Overall, this research contributed a novel approach to adaptive malware detection to potentially enhance future malware detection systems in the face of evolving cyber threats. Future work will expand the scope of experimentation to account for various types of malware and underlying operating systems. The current focus is competitively evolving increasingly robust malware classifiers in response to increasingly sophisticated synthetic malware, where classifiers and synthetic malware are co-evolved from a broad range of malware datasets. An end goal is to automate the adaptation of anti-malware classifiers in response to constantly emerging malware, as part of larger research effort to produce perpetually adapting and self-sustaining autonomous systems [21].

References

[1] D. Gibert, C. Mateu, and J. Planes, “The Rise of Machine Learning for Detection and Classification of Malware: Research Developments, Trends and Challenges,” *Journal of Network and Computer Applications*, vol. 153, p. 102526, 2020.

[2] M. Gopinath and S. Sethuraman, “A Comprehensive Survey on Deep Learning based Malware Detection Techniques,” *Computer Science Review*, vol. 47, p. 100529, 2023.

[3] R. Lo *et al.*, “MCF: A Malicious Code Filter,” *Computers Security*, vol. 14, no. 6, pp. 541–566, 1995.

[4] K. Rieck *et al.*, “Automatic Analysis of Malware Behavior using Machine Learning,” *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[5] X. Ling *et al.*, “Adversarial attacks against Windows PE Malware Detection: A Survey of the State-of-the-Art,” *Computers Security*, vol. 128, p. 103134, 2023.

[6] T. Alsmadi and N. Alqudah, “A Survey on Malware Detection Techniques,” in *2021 International Conference on Information Technology (ICIT)*, pp. 371–376, 2021.

[7] Y. LeCun, Y. Bengio, and G. Hinton, “Deep Learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[8] K. Aryal, M. Gupta, and M. Abdelsalam, “A Survey on Adversarial Attacks for Malware Analysis,” 2022.

[9] F. Pierazzi *et al.*, “Intriguing Properties of Adversarial ML Attacks in the Problem Space,” in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 1332–1349, IEEE, 2020.

[10] X. Zeng *et al.*, “Adversarial Attacks Beyond the Image Space,” 2019.

[11] Anon, “Anonymous Repository,” *https://github.com/neatcnn/malwaredetection*, 2024.

[12] K. Stanley and R. Miikkulainen, “Competitive Coevolution through Evolutionary Complexification,” *Journal of Artificial Intelligence Research*, vol. 21, pp. 63–100, 2004.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, USA: MIT Press, 2016.

[14] A. Gaier and D. Ha, “Weight Agnostic Neural Networks,” *arXiv*, vol. 1906.04358, 2019.

[15] A. Kumar, K. Kuppusamy, and G. Aghila, “A Learning Model to Detect Maliciousness of Portable Executable using Integrated Feature Set,” *Journal of King Saud University - Computer and Information Sciences*, vol. 31, no. 2, pp. 252–265, 2019.

[16] B. Chen *et al.*, “Adversarial examples for cnn-based malware detectors,” *IEEE Access*, pp. 54360–54371, 2019.

[17] T. Wang, , C. Wu, and C. Hsieh, “Detecting Unknown Malicious Executables Using Portable Executable Headers,” in *Proceedings of the Fifth International Joint Conference on INC, IMS and IDC*, pp. 278–284, 2009.

[18] B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes*. Cambridge, UK: Cambridge University Press, 1986.

[19] S. Sen, E. Aydogan, and A. Aysan, “Coevolution of mobile malware and anti-malware,” *IEEE Transactions on Information Forensics and Security*, p. DOI:10.1109/TIFS.2018.2824250, 2018.

[20] K. Babaagba and J. Wylie, “An Evolutionary based Generative Adversarial Network Inspired Approach to Defeating Metamorphic Malware,” in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pp. 1753–1759, 2023.

[21] G. Nitschke and D. Howard, “Autofac: The perpetual robot machine,” *IEEE Transactions on Artificial Intelligence*, vol. 3, no. 1, pp. 2–10, 2022.