Malware detection through low-level features and stacked denoising autoencoders

Alessandra De Paola[1], Salvatore Favaloro[2], Salvatore Gaglio[1], Giuseppe Lo Re[1], and Marco Morana[1]

[1] Università degli Studi di Palermo
firstname.lastname@unipa.it
[2] firstname.lastname@community.unipa.it

Abstract

In recent years, the diffusion of malicious software through various channels has gained the request for intelligent techniques capable of timely detecting new malware spread. In this work, we focus on the application of Deep Learning methods for malware detection, by evaluating their effectiveness when malware are represented by high-level, and low-level features respectively. Experimental results show that, when using high-level features, deep neural networks do not significantly improve the overall detection accuracy. On the other hand, when low-level features, i.e., small pieces of information extracted through a light processing, are chosen, they allow to increase the capability of correctly classifying malware.

1 Introduction

Malware detection is one of the most critical issues faced by computer security. Nowadays, the accidental execution of malicious software coming from different channels makes IT systems constantly exposed to risks. In order to effectively detect threats hidden behind heterogeneous software, which is susceptible to unpredictable variations, intelligent techniques are required.

The most common approach implemented by popular malware-detection tools is to perform a static analysis of a short sequence of bytes, i.e., a signature [24]. Nevertheless, the availability of softwares allowing the automatic generation of several variants of a certain malware greatly reduces the effectiveness of this technique [23].

One of the most promising direction followed to guarantee a high detection rate even with the constantly increase of threats, is the adoption of cloud-based approaches where the malware detection is performed remotely through machine learning techniques capable of analyzing a huge amount of malware files. The effectiveness of this solution often relies on the adoption of high-level features, extracted from executable files, designed to be more invariant to code obfuscation and polymorphism than classic signature-based methods.

Nevertheless, it is worth noticing that designing "good" set of features to realize efficient and effective systems is not trivial, due to the underlying and not explicit dependences among different parts of a file and its behavior. A promising way to face such a problem is represented by deep learning, which is expected to enable the automatic extraction of relevant features, directly from raw information.

Although the potentiality of applying deep learning on simple raw data is confirmed by some recent results, many works adopt deep learning to perform malware detection or classification on high-level features obtained from executable files.

The first goal of the proposed work is to evaluate the real effectiveness of malware detection through deep learning, while processing high-level features. To this aim, we compare the system proposed in [16] with other classifiers of various complexity.

Moreover, we want to verify if deep networks can be effectively adopted to perform malware detection by exploiting a reduced set of low-level features extracted from the executable files. The solution we propose here is based on a deep network implemented through stacked denoising autoencoders, which analyzes raw information contained in the portable executable (PE) packaging of Windows executable files. Our aim is to compare the performance obtained by such lighter malware detection system with the most complex approach proposed in [16], in order to establish whether deep learning is able to achieve good performance even with fewer and less pre-processed data.

The following of the paper is structured as follow. Sect. 2 presents an overview of the related work. Sect. 3 describes the high-level features and the system proposed in [16], besides the simpler classifiers we use for the comparative analysis. Sect. 4 provides a description of the proposed malware detection system based on low-level features. Finally, Sect. 5 reports the experimental evaluation, and Sect. 6 presents our conclusions and possible future work.

2 Related Work

Several works which adopt deep learning to perform malware detection have been presented in the recent literature, based both on dynamic and static analysis [10]. The dynamic analysis exploits a protected environment, i.e., a "sandbox", in which the malware can be executed, and its behavior observed, without threaten a real working system, while the static analysis focuses only on information included in the target file, and do not require that the potential malware is run. As a consequence, static methods are generally faster and less greedy for resources, even if they can exploit less information and are vulnerable to malicious code obfuscation.

The system proposed in [5] performs a dynamic analysis, by adopting deep learning to automatically generate the signature which represents such behavior. The authors use a deep belief network implemented through stacked denoising autoencoders, which processes the text file containing the transcription of all the events occurred during a file run, adequately converted in a binary form. The automatically-generated signature is then processed by a SVM to perform the effective classification. In [8], the sequence of system calls recorded while the executable file is running is exploited by a Deep Neural Network (DNN) which combines convolutional layers and recurrent layers, using Long Short Term Memory (LSTM) cells to increase malware detection capabilities.

In order to overcome the limitation of dealing with less information than those obtained by dynamic analysis, some works based on static analysis performs deep learning on high-level features extracted according to a well-designed process. The authors of [4] propose to process a file to build a large vector of features (179 thousands of sparse binary features), which is reduced through a random projection process. The resulting vector, composed of 4,000 elements, is then analyzed by a deep classifier which is pre-trained through a Restricted Bolzmann Machines (RBM). The reduction of the feature vectors through random projection is also adopted in [20], where JavaScript sources are processed through a 5–layer deep neural network implemented with stacked denoising autoencoders. In [16] four different sets of static features, converted in a 1024-length binary vectors and classified by a classic DNN, are used. A more complex set of features is propose in [22], where data is obtained by merging information coming from static and dynamic analysis. Such set of features includes a sub-set automatically obtained by DNNs. The resulting advanced set of features is classified through Multiple Kernel Learning.

Even though malware detection algorithms which combine deep learning and high-level features usually provide better accuracy values, some authors emphasize the convenience to adopt features as simple as possible so as to design light and efficient malware-detection systems [2].

The potentiality of applying deep learning on simple raw data is confirmed by some works recently presented in the literature, such as [11], where an android malware detection system which applies a deep convolutional neural network to the raw sequences of opcode extracted from disassembled programs is presented.

Our work aims to confirm this idea, by proving that deep learning can be efficiently adopted to perform malware detection when executable files are represented by simple features.

3 High-level features for malware detection

Over the years, several type of features have been adopted for malware detection through static analysis. Some of them are obtained from a heavy pre-processing of the executable file to summarize its global characteristics, or through a simple and light pre-processing phase. The former can be considered high-level features, since they allow to represent the executable file at a higher level of abstraction. The latter can be defined low-level features, since they are closer to the raw file representation.

Authors of [13] proved the effectiveness of several machine learning methods using high-level features for distinguishing between packed and non-packed executables. This work adopts a small set of features which summarize high-level aspects of the analyzed software, i.e., the number of standard and non-standard sections in the portable executable (PE) packaging of a Windows executable file, the number of executable sections, the number of readable/writable/executable sections, the number of entries in the Import Address Table, and the entropy of PE header, code section, data section, and of the whole executable file. The authors of [15] analyze the frequencies of opcode sequences classifying them through Support Vector Machines. Some works focus on the analysis of the frequency of Windows API calls, such as [3, 25, 21]. In [17], sequences of strings and bytes extracted from the file are considered together with the set of API calls. Authors of [1] propose to use n-grams of strings, whereas in [12] and [4] n-grams based on n-length system calls sequences are used. Authors of [2] propose to merge a large set of complex features by selecting the best subset of features through a forward stepwise selection. The set of features includes n-grams of bytes, the byte-level entropy vector, some descriptors of the image obtained by interpreting each byte as a grey level, the histogram of the length of strings, the frequency of some specific operation codes, the frequency of API calls, together with many other features.

One of the goal of our work is to verity whether, still using high-level features, deep learning can further improve the detection accuracy. As target system for our analysis, we consider the work proposed in [16], which adopts a deep neural network to classify vectors of high-level features extracted from executable files. The feature vector is obtained by merging four complementary histograms, which depend on different information extracted from the analyzed file. Two mono-dimensional histograms capture information from the PE, while the other two bi-dimensional histograms depend on the whole file. The concatenation of these four histograms, after the row-by-row concatenation of the bi-dimensional components, produces a single mono-dimensional vector composed by 1024 elements. The first mono-dimensional part of the feature vectors is the *PE import histogram*, obtained by reading the *import address table* header of PE and generating the tuples of DLLs and functions listed in the header. Each tuple is mapped in the corresponding hash value and the component is computed as the histogram of such hash values. The second mono-dimensional part is the *PE metadata histogram*, which depends on labels and values of numerical fields contained in the PE packaging. Each label-value pair is mapped in the corresponding hash value and the histogram of the hash values is computed. The bi-dimensional components of the feature vector are the *byte/entropy histogram* and the

string histogram. The former is created by sliding a 1024-byte-sized window over the file with step of 256 bytes, and computing the base-2 entropy for each window. For each byte in the window, a byte-entropy pair is computed; then, the set of pairs is processed to create a 16x16 byte/entropy matrix, i.e., the final 2D histogram. The string histogram is obtained by extracting the file strings long at least 6 characters. Each string is processed to obtain a pair containing the hash value of the string itself and the logarithm of its length. All the resulting pairs are considered to obtain a 16x16 hash/length histogram.

The malware detection process is based on a deep neural network composed by four layers: an input layer consisting of 1024 nodes, two hidden layers, each consisting of 1024 nodes, and a final layer with a single node. Nodes of the two hidden layers adopt the Parametric Rectified Linear Unit (PReLU) [6] as activation functions, whereas activation of the node in the final layer depends on a sigmoid. PReLU is an activation function, recently proposed in the literature, which autonomously modifies its form according to a specific parameters that can be changed to speed up the training phase. Training is performed by means of a back-propagation algorithm and the Adam gradient-based optimizer [7], a recent stochastic optimizer that uses the first and second moments of the gradient to minimize the objective function, i.e., the binary cross-entropy. Finally, to prevent overfitting, *dropout* regularization [18] is performed on the first three layers. The dropout technique disables some units of the neural network during the training so as to process a narrow sub-graph rather than the whole network. On every learning round, different units are randomly sampled to be disabled, so guaranteeing that the training is performed on different sub-networks.

Sect. 5 presents the comparative analysis of such target system with other classifiers of decreasing complexity, which adopt the same set of high-level features. Surprisingly, the results we obtained prove that with this well-designed set of high-features, deep learning does not have a relevant impact on detection accuracy as compared with other simpler classifiers.

4 Low-level features and DNNs for malware detection

Rather than exploiting high-level features, we propose to adopt deep networks to process low-level features extracted through static analysis, in order to automatically learn which information is relevant to malware detection. The system we propose is based on a two-phase training deep neural network, where a first unsupervised pre-training with stacked denoising autoencoders [19] is followed by a supervised fine-tuning based on back-propagation.

Features extraction is performed by accessing *DOS Header*, *File Header*, *Optional Header* and *Section Table* of the PE packaging. Each header contains different fields, each analyzed by extracting the corresponding value and two offsets. Simple values are handled as unsigned integers, whilst timestamps, arrays and strings, are processed via a hashing function. Offset values allow to preserve spatial information: the *local offset* specifies the position of a field within the header section, and the *global offset* represents the position in the file.

Since a file can have a variabile number of sections and the classifier is designed to process fixed-size data, we limit the number of *Section Tables* to be processed. To this aim, some experiments were performed on real data showing that a reasonable threshold on the number of sections is 13. If a file contains less sections than the threshold then the elements of the feature vector related to the missing sections are set to zero; otherwise, if the number of sections is higher than the threshold, the extra sections will be ignored. Thus, 19 fields of the DOS header, 7 of the file header, 30 of the optional header, 12 of the section section header (for each of the 13 sections) were analysed, obtaining feature vectors of 636 elements in the range $[0, 1]$. These vectors are about 38% smaller than those proposed in [16], and described in Sect. 3. Moreover,

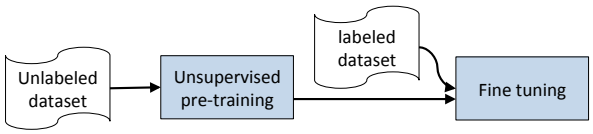


Figure 1: Two-phase training.

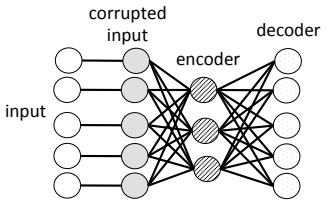


Figure 2: Structure of the denoising autoencoder.

since data are not extracted from the entire file but only from its headers, the feature extraction process is fast and independent of the file size.

The deep network proposed here is consists of five layers. The input layer has 636 elements, the three hidden layers contain 256, 64, and 16 nodes respectively, whilst the output layer consists of a single node. Each node use a sigmoid activation function. In this neural network, the total amount of trainable hyper-parameters is of $180,577$ elements.

The training of the deep network is performed in two steps (see Fig.1). The first phase performs an unsupervised pre-training by using unlabeled dataset to obtain a first estimation of weights and biases of the hidden layers. Such phase is implemented through stacked denoising autoencoders. Each hidden layer is pre-trained individually by means of a support network consisting of an input layer, a corrupted layer, an encoder, and a decoder layer (see Fig. 2), where the encoder layer corresponds to the hidden layer to be pre-trained. On the contrary, the input, the corrupted, and the decoder layers have the same number of nodes, which is equal to the number of inputs of the hidden layer to be pre-trained. This support network is trained through back-propagation and processes each input vector by i) corrupting it with some kind of noise, ii) encoding the resulting noisy signal, and then iii) reconstructing the encoded signal. The aim of the corruption layer, which adds isotropic Gaussian noise to the input signals, is to force the encoding layer to learn the most useful information from input vectors, by automatically neglecting noise from the corrupted input. The iteration of the pre-training for all the hidden layers produces a deep neural network in which each layer is able to extract and represent features at a higher level of abstraction than its predecessors. The pre-training of the first hidden layer is performed using the original dataset, while the other hidden layers are pre-trained through an encoded version of the dataset, obtained by exploiting the current trained denoising autoencoders to build a temporary network which extracts the output of the encoding layer that precedes the layer to be trained.

The second training step is the fine-tuning of the network, which is implemented through a supervised back-propagation algorithm with the Adam stochastic optimization, applied by considering the binary cross entropy as objective function. During this stage, weights and bias of all hidden layers are initialized with the values produced by the pre-training.

5 Experimental Evaluation

The dataset used for the experimental evaluation was obtained by merging 12.000 samples of malware obtained from VirusShare [1] and 11.874 samples of certified software obtained from a clean Windows 10 installation. The malware samples are not labeled according to the specific

[1] https://virusshare.com

Classifier	Input	Hidden Layer 1	Hidden layer 2	Output	n. parameters
Benchmark [16]	1,024	1,024	1,024	1	2,102,273
Shallow	1,024	2,048	0	1	2,103,297
Classic-1024	1,024	1,024	0	1	1,050,625
Classic-512	1,024	512	0	1	525,313
Classic-8	1,024	8	0	1	8,209

Table 1: Number of nodes composing each layer of the analyzed neural networks. The last column reports the number of trainable parameters.

malware family they belong to, but this characteristic does not limit the experimental evaluation presented here, since it aims only to assess the detection accuracy.

The evaluation of each classifier is performed through a K-fold cross validation with a stratified sampling, in order to preserve the percentage of samples per class. Due to the limited size of the dataset used here, we adopted $K = 5$ for each test so as to guarantee that each validation set contains an adequate number of samples. Each model was trained until the loss value dropped below 0.02, or the number of training epochs exceeded 200, as proposed in [16].

The performance of each classifier were evaluated by analyzing the trend of the ROC curve (Receiver Operating Characteristic) with respect to the training epochs. Furthermore we computed the final loss, the final accuracy, as well as several other metrics, i.e., TPR (True Positive Rate), FPR (False Positive Rate), Precision, and AUC (Area Under Curve) of ROC curve. This last metrics allows to evaluate the classification performance independently on the threshold adopted for the last layer.

5.1 Deep Learning performance with high-level features

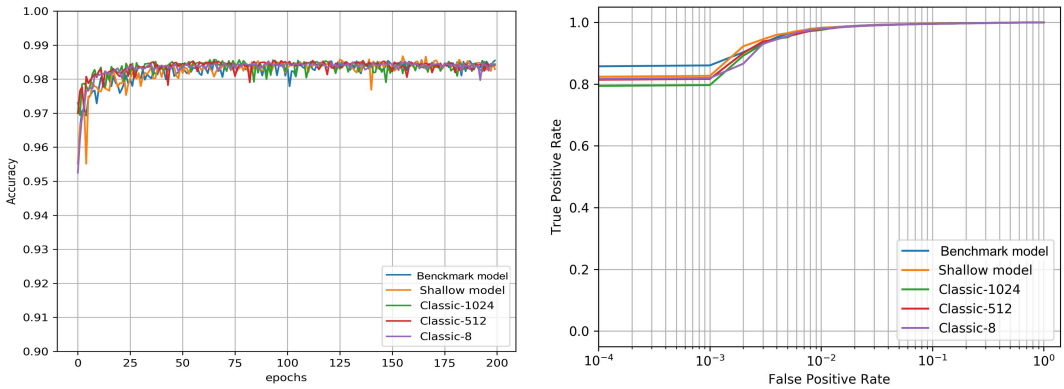
In order to verify the impact of deep learning methods when high-level features are used to represent files, we performed a comparative evaluation between the benchmark described in Sect. 3, and different classifiers based on neural networks, which adopt the same set of features.

The first neural network involved in the comparison, named *shallow classifier*, is obtained by merging the two hidden layers of the benchmark network into a single double-sized layer. The shallow network differs from the benchmark only for its topology, since it has almost the same numbers of parameters. The comparison between the benchmark and the shallow networks allows to evaluate the effect of the hierarchical stratification on the performances of the classifier. The second network is obtained by removing the second hidden layer from the benchmark network, by replacing the PReLUs with sigmoids, and by removing the dropout step, thus obtaining a *classic* neural network. Then, by varying the number of nodes contained in the remaining hidden layer, we obtained three neural networks with 1024, 512 and 8 nodes in the hidden layer. Such networks have less parameters than the benchmark, and their evaluation allows to analyze how much deep learning techniques can improve the classification accuracy, when high-level features are adopted. Table 1 summarizes the topology and the number of parameters for each network.

Results are summarized in Table 2. Surprisingly, even if the benchmark network allows to achieve the best results for almost every metric, the results obtained by the other classifiers are quite similar, even those obtained by the classic network with only 8 nodes on the hidden layer. Moreover, by analyzing the trend of the average accuracy with respect to the training epochs (Fig. 3a) and the ROC curves (Fig. 3b), no relevant difference among the considered neural networks arises. Such results confirm that, if a set of well-designed high-level features

Classifier	Loss	Accuracy	Precision	TPR	FPR	AUC
Benchmark [16]	**0.0707**	**98.55**	98.66	**98.45**	1.20	**99.76809**
Shallow	0.0931	98.29	**99.07**	97.53	**0.93**	99.75425
Classic-1024	0.0926	98.44	98.60	98.30	1.41	99.72532
Classic-512	0.0858	98.40	98.66	98.17	1.36	99.70791
Classic-8	0.0770	98.41	98.87	97.96	1.13	99.73027

Table 2: Experimental evaluation of different neural networks using high-level features as input.



(a) (b)

Figure 3: Average accuracy (a) and ROC curves (b) of different networks using HL features.

is adopted to represent malware files, deep neural networks do not significantly impact on the performances.

5.2 Deep Learning performance with low-level features

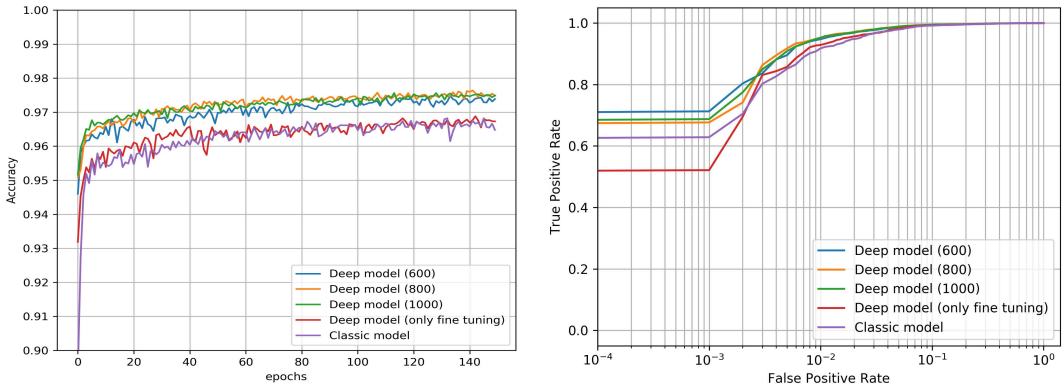
Several tests were performed to evaluate both the effectiveness of the proposed approach, based on the adoption of deep learning with low-level features, and the impact of the design choices we made.

Firstly, we wanted to investigate whether there is a connection between the number of training epochs and performance. For this purpose we compared the performance obtained by varying the maximum training epoch threshold (i.e., 600, 800, and 1000) for the pre-training phase. The second evaluation aimed to assess the effectiveness of the pre-training phase, by comparing our system with a deep network characterized by the same topology, but trained only through the second training phase, that is without pre-training. Finally, we intended to verify the impact of the hierarchical stratification of our deep classifier on the overall performances of the malware detection system. To this aim, we compared our system with a classic neural network obtained by removing the last two hidden layers from our classifier, so obtaining three layer of 636, 256 and 1 nodes respectively.

In Table 3 the average values of the considered performance metrics are shown. We can observe that higher accuracy values are obtained by using the pre-trained classifiers, and this confirms the effectiveness of the two-phase training. However, increasing the number of pre-training epochs (e.g., from 600 to 800 or 1000) does not significantly improve the accuracy, nor reduce the values of loss.

Classifier	**Loss**	**Accuracy** (%)	**Precision** (%)	**TPR** (%)	**FPR** (%)	**AUC** (%)
Deep model (600)	0.075	97.39	97.48	**97.33**	2.55	97.39
Deep model (800)	**0.074**	**97.48**	**98.09**	96.88	**1.91**	97.48
Deep model (1000)	**0.074**	**97.49**	97.92	97.08	2.09	**97.49**
Deep model (*fine tuning* only)	0.087	96.73	96.69	96.82	3.37	96.73
Classic-623-256	0.095	96.48	95.98	97.10	4.15	96.47

Table 3: Average values Loss, Accuracy, Precision, TPR (True Positive Rate), FPR (False Positive Rate), and AUC (Area Under Curve) using different classifiers. The number of pre-training epochs is reported in parentheses.



(a) (b)

Figure 4: Average accuracy (a) and ROC curves (b) of different models. Pre-train epochs are reported in parentheses.

The performances of the model that uses fine tuning only, i.e., trained only with the second training phase, are worse than the pre-trained models on all metrics. Furthermore, the lowest values are achieved by using the classic model, suggesting that the hierarchical stratification is effective, but only when combined with the pre-training phase.

A more in-depth analysis of the average accuracy achieved using different models is shown in Fig. 4a, which uses results from the second training phase. Results confirm that the pre-trained deep models exhibit a better trend along all the time line. Fig. 4b shows the average ROC curves from which AUC score was calculated. Even in this case, pre-trained models outperform the others, with no significative differences when using 800 or 1000 pre-training epochs.

When adopting low-level features, our deep model produces an increase of more than 2% of precision and of 1% for accuracy with respect to a traditional neural network. On the contrary, the adoption of a deep network in the benchmark system increase the accuracy and precision values of only 0.06% and 0.11% respectively.

It is worth noticing that, whatever is the classifier, the adoption of *optimal* high-level features yields to better classification performances. Thus, even though using a smaller and simpler set of features makes the malware detection process easier, the proposed deep learning solution is not able to achieve the same level of accuracy.

6 Conclusions and Future Work

In this work we addressed the scenario of malware detection through deep learning approaches. Firstly, we wondered if deep learning can makes a relevant difference when combined with well-designed high-level features. Experiments proved that, when using an optimal set of features, the adoption of a deep neural network does not significantly influence the performance.

Since deep learning is often adopted to achieve a high classification accuracy even by processing raw data, we also investigated whether the use of low-level features (computed only on the first 1 or 2 KB of a file) combined with a deep learning approach may lead to comparable performances. We proposed a deep neural network whose parameters are learned through a two-phase training which combines a unsupervised technique, i.e., stacked denoising autoencoders, with a supervised step to perform fine tuning.

Results showed the capability of our system of extracting relevant knowledge from low-level features obtained through a static analysis of the considered files. Moreover, the two-phase training was proved to be very effective regardless of the number of pre-training epochs.

However, regardless of the classifier, the use of *optimal* high-level features leads to the best classification performances. Nevertheless, it is worth noticing that with a total of 180,577 parameters, the 9% of those used by the benchmark system [16], our system exhibits a reduction of only 0.57% and 1.07% in precision and accuracy respectively.

Such results confirm the potentiality of our approach, and also highlight the necessity of a further update of the method we propose in order to overcome the limitation introduced by adopting a smaller and simpler set of features.

As future work, we want to verify if other classifier can be adopted for the first, unsupervised, training phase. For example a noisy-based autoencoder, like the DA-IC (*Denoising Autoencoder with Interdependent Codes*) [9], or a deterministic one, like the *contractive autoencoder* [14] could lead to better performances.

References

[1] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan. N-gram-based detection of new malicious code. In *Proc. of the 28th Annual Computer Software and Applications Conf. (COMPSAC 2004)*, volume 2, pages 41–42. IEEE, 2004.

[2] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto. Novel feature extraction, selection and fusion for effective malware family classification. In *Proc. of the Sixth ACM Conf. on Data and Application Security and Privacy*, pages 183–194. ACM, 2016.

[3] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab. Zero-day malware detection based on supervised learning algorithms of api call signatures. In *Proc. of the Ninth Australasian Data Mining Conf. - Volume 121*, AusDM '11, pages 171–182, 2011.

[4] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu. Large-scale malware classification using random projections and neural networks. In *Proc. of the 2013 IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3422–3426. IEEE, 2013.

[5] O. E. David and N. S. Netanyahu. Deepsign: Deep learning for automatic malware signature generation and classification. In *Proc. of the 2015 Int. Joint Conf. on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. of the IEEE Int. Conf. on computer vision*, pages 1026–1034, 2015.

[7] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proc. of the 3rd Int. Conf. on Learning Representations (ICLR 2015)*, pages 1–15, 2014.

[8] B. Kolosnjaji, A. Zarras, G. D. Webster, and C. Eckert. Deep learning for classification of malware system call sequences. In *Proc. of the Australasian Conf. on Artificial Intelligence*, volume 9992 of *Lecture Notes in Computer Science*, pages 137–149. Springer, 2016.

[9] H. Larochelle, D. Erhan, and P. Vincent. Deep learning using robust interdependent codes. In *AISTATS*, pages 312–319, 2009.

[10] K. Mathur and S. Hiranwal. A survey on techniques in detection and analyzing malware executables. *Int. J. of Advanced Research in Computer Science and Software Engineering*, 3(4), 2013.

[11] N. McLaughlin, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupé, and G. Joon Ahn. Deep android malware detection. In *Proc. of the Seventh ACM on Conf. on Data and Application Security and Privacy*, CODASPY '17, pages 301–308. ACM, 2017.

[12] S. B. Mehdi, A. K. Tanwani, and M. Farooq. Imad: in-execution malware analysis and detection. In *Proc. of the 11th Annual Conf. on Genetic and evolutionary computation*, pages 1553–1560. ACM, 2009.

[13] Roberto Perdisci, Andrea Lanzi, and Wenke Lee. Classification of packed executables for accurate computer virus detection. *Pattern Recognition Letters*, 29(14):1941 – 1946, 2008.

[14] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proc. of the 28th Int. Conf. on machine learning (ICML-11)*, pages 833–840, 2011.

[15] I. Santos, F. Brezo, B. Sanz, C. Laorden, and P. G. Bringas. Using opcode sequences in single-class learning to detect unknown malware. *IET information security*, 5(4):220–227, 2011.

[16] J. Saxe and K. Berlin. Deep neural network based malware detection using two dimensional binary program features. In *Proc. of the 2015 10th Int. Conf. on Malicious and Unwanted Software (MALWARE)*, pages 11–20. IEEE, 2015.

[17] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. Data mining methods for detection of new malicious executables. In *IEEE Symposium on Security and Privacy*, pages 38–49. IEEE Computer Society, 2001.

[18] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. of Machine Learning Research*, 15(1):1929–1958, 2014.

[19] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. of Machine Learning Research*, 11(Dec):3371–3408, 2010.

[20] Y. Wang, W.-D. Cai, and P.. Wei. A deep learning approach for detecting malicious javascript code. *Security and Communication Networks*, 9(11):1520–1534, 2016.

[21] M. Weber, M. Schmid, M. Schatz, and D. Geyer. A toolkit for detecting and analyzing malicious software. In *Proc. of the 18th Annual Computer Security Applications Conf.*, pages 423–431. IEEE, 2002.

[22] L. Xu, D. Zhang, N. Jayasena, and J. Cavazos. Hadm: Hybrid analysis for detection of malware. In *Proc. of the SAI Intelligent Systems Conf. (IntelliSys)*, pages 1037–1047, 2016.

[23] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar. A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)*, 50(3):41, 2017.

[24] Y. Ye, T. Li, S. Zhu, W. Zhuang, E. Tas, U. Gupta, and M. Abdulhayoglu. Combining file content and file relations for cloud based malware detection. In *Proc. of the 17th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 222–230, 2011.

[25] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang. An intelligent pe-malware detection system based on association mining. *J. in computer virology*, 4(4):323–334, 2008.