Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 159 (2019) 182–191

Procedia Computer Science

www.elsevier.com/locate/procedia

23rd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

A Model Based on LSTM Neural Networks to Identify Five Different Types of Malware

Eduardo de O. Andrade[a], José Viterbo[a], Cristina N. Vasconcelos[a], Joris Guérin[a], Flavia Cristina Bernardini[a]

[a]*Universidade Federal Fluminense, Gal. Milton Tavares de Souza Av., Niterói-RJ 24210-346, Brazil*

Abstract

Identifying malware has always been a great challenge. Much money and time has been invested by companies and governments to mitigate the impact of these threats. Nowadays, with the increasing amount of data available, it is possible to use more precise classification techniques. However, most large datasets that include malicious and non-malicious softwares are not public, which hinders the quest for solutions based in technologies that rely on the availability of large amounts of data, such as deep learning. To overcome this limitation, this article introduces a new large dataset for malware classification, which was made publicly available. We then propose a model to train a multiclass classification recurrent neural network (RNN), more specifically a long short-term memory neural network (LSTM) on our dataset. This model for analyzing unstructured malware data is then tested on unseen programs and the accuracy obtained reaches 67.60%, including six classes with five different types of malware.

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

1. Introduction

The number of new malwares (short for malicious softwares) that emerge every moment continues to grow at high rates every year [18]. The interest of companies and governments on the subject is notorious, since cyber attacks that include the spread of malware can have strongly negative impacts on finances and businesses [21]. Therefore, we can note a steep increase in the amount of data about these threats being made available. However, the analysis of such data is complex, requiring multiple studies to try and determine what best characterizes and differentiates a cleanware (short for clean software) file from a malware file [5].

* Eduardo de O. Andrade. Tel.: +55-21-98179-2699.
E-mail address: eandrade@ic.uff.br

1877-0509
10.1016/j.procs.2019.09.173

The machine learning area has been promising in terms of malware classification [7]. When a large amount of data is available, deep learning, which is a machine learning subarea, enables to achieve better results than traditional approaches [9], using artificial neural networks (ANNs) models, such as recurrent neural networks (RNNs). The initial use of these networks was mainly image classification, but this has changed with time and today there are several areas where deep learning can be applied. Nevertheless, to achieve this purpose, we need to deal with different characteristics present in each domain.

One of these domains where the application of deep learning and the RNNs has been widely used is language modeling [20]. The states of these networks aid in the understanding of sentences, in which so-called "cells", in the case of long short-term memory neural networks (LSTMs), process one word at a time and return the probabilities of possible values that the next word may take in a sentence. The "deep" aspect of the RNNs represents the transformation process that such data suffers, which becomes more abstract as they reach higher levels in the networks. In this way, some details of the data turn out to be more relevant, while other details become less important and are disregarded.

In this work, we try to explore the understanding of the "language" of malwares by looking at their hexadecimal codes as sentences. We follow the approach proposed by [14] with malwares obtained from VirusShare[1], which is a repository of malwares taken from a variety of Windows systems. The *hexdump* tool generate text files with hexadecimal representation that are used as input into our LSTM model.

Our malware classification problem modeling was made specifically for the dataset we created and made publicly available. We believe that the lack of adequate datasets is a major problem for studies involving malware analysis and other threats. In our case, it was very difficult to find open datasets with a great number of cleanware samples available and properly labeled. So, we made a labeling with several classes, including five different types of malware for our dataset. The motivation behind this article also includes this use of a good publicly available dataset for malware classification and the objective of this work is to obtain good classification results with the malware types included into MC-dataset-multiclass.

This paper is organized as follows. In Section 2, we provide a brief description of malware types, analysis and RNN. In Section 3, we discuss the related work. In Section 4, we present our dataset. In Section 5, we present our LSTM model. In Section 6, we describe our experiment and discuss about the obtained results. Finally, in Section 7, we draw our conclusions and propose future work.

2. Background

In this section we present the different malware types included in our dataset and the current methodologies to deal with malware analysis. We also give an overview of the concept of RNNs, especially LSTMs, which are used in this work.

2.1. Malware Types

Malware refers to any software that was intentionally written to cause damage to a computer, mobile device, data or even people. Five types of malware were included in our dataset. Each type of malware has its own characteristics and malwares can also have more than one type.

Backdoor. It is essentially designed to bypass authentications procedures, compromissing the system. After the bypass, more backdoors may be installed, allowing future access for the attacker without being detected by the user.

Rootkit. This malware is implemented to be "invisible" for the users and security softwares. It tries to take control of a computer or gain remote access in this way.

Trojan. Trojan horse or trojan is probably the most common type of malware. This malware, through a "camouflage", tries to pass as a conventional cleanware or some program that the user is convinced to download. Once this is done, unauthorized access is granted to the computer on which it is installed and can be used to monitor user activities, steal data, modify files, make the computer part of a botnet, and so on.

Virus. The virus is capable of making multiple copies of itself. For example, it can spread on several computers connected to the same network, and is activated by running an infected software that the user does not know.

[1] https://virusshare.com

Worm. One of the most dangerous types of malware, the worm is able to spread quickly through a network of computers by exploiting operating system vulnerabilities. The worm is also able to replicate itself but unlike the virus, it does not need a program execution by the user to occur.

2.2. Malware Analysis

Basically, we have two types of malware analysis: dynamic (also called behavioral analysis) and static (also called code analysis). Dynamic analysis usually consists of monitoring the behavior of malware by running them in a controlled environment, such as a *sandbox*, preventing a system infection. Static analysis does not involve malware execution. It seeks to extract features of the binary files and also can be carried out processes of disassembly [6]. This article focuses on the problem of static malware analysis.

An antivirus software usually performs static analysis type of approach. Signatures of the malwares are collected and added in the antivirus database. This procedure usually requires less computing power. In spite of this, several techniques present in the current malwares try to circumvent this verification by the antivirus, such as metamorphism and polymorphism [22].

In the malware identification process, whether by dynamic or static analysis, the accuracy is not the only important metric to optimize. Indeed, false negatives and false positives are also important. Classifying a malware as a cleanware is apparently the most dangerous case, but classifying a cleanware as malware can also be harmful to businesses and governments in financial terms.

2.3. Recurrent Neural Network

The main feature that differentiates an RNN from other neural networks is the formation of direct cycles between their neurons. This provides the network with temporal processing and sequence learning by creating internal states. These details and the complexity of the RNNs usually cause them to demand a greater amount of memory and also a higher execution time.

These internal states in the RNNs make it maintain part of the activation of the previous inputs and do a "feedback" of the calculations of the network with each step of time. For example, for a single hidden layer RNN, we define $\boldsymbol{t}$ as the t-th step of time, $\boldsymbol{i(t)}$ and $\boldsymbol{o(t)}$ as the input and output vectors respectively, $\boldsymbol{h(t)}$ as the hidden vector corresponding to the hidden layer, $\boldsymbol{f}_h$ and $\boldsymbol{f}_o$ as the activation functions of the hidden layer and output layer respectively, $\boldsymbol{P}_{ih}$ as the weight matrix between the input layer and the hidden layer, $\boldsymbol{P}_{hh}$ as the weight matrix between the hidden layer and the own hidden layer and $\boldsymbol{P}_{ho}$ as the weight matrix between the hidden layer and the output layer, we have the Equation 1, expressing the hidden layer activation and Equation 2, returning the outputs.

$$h(t) = f_h(i(t)P_{ih} + h(t-1)P_{hh}) \tag{1}$$

$$o(t) = f_o(h(t)P_{ho}) \tag{2}$$

The LSTM neural network is a subtype of RNN that seeks to solve the problem of "long-term dependence" [11]. The memory cells present in the LSTMs were implemented to circumvent this problem because they are able to preserve states against long time periods. We can observe the basic architecture of an LSTM memory block or unit inn Figure 1. We can have several units of these that connect to a standard structure of an RNN or another units.

Each unit in the LSTM contains memory cells that, through self connections, store the network time state. Also in each unit there are 3 gates, the forget gate, the input gate and the output gate. The forget gate acts using the internal state of the cell before adding it again to the cell as input. This happens using self recurrent connection, resetting the

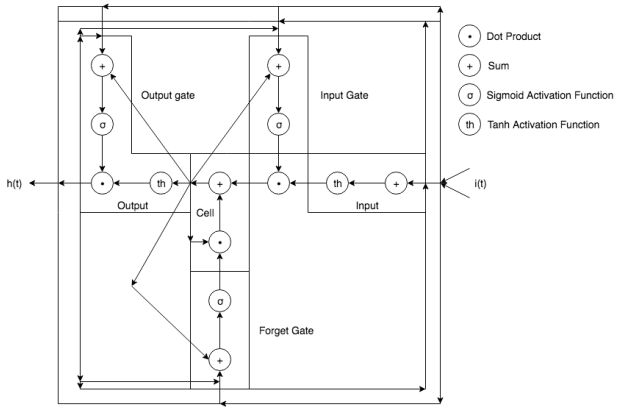

Fig. 1. LSTM architecture. The neural network is usually composed by many memory blocks

cells memory in an adaptive way. The input gate tries to control the input activations in relation to the memory cell. Controlling the output of cell activations is the responsibility of the output gate. Lastly, the peephole connections (the connections that leave the memory cell to the gates) it is an attempt to learn precise timing of the outputs but they do not increase the performance by much and are sometimes omitted on LSTM's units representations [3].

Defining σ as the logistic function, $\boldsymbol{y}$, $\boldsymbol{x}$, $\boldsymbol{w}$ and $\boldsymbol{z}$ as the forget gate, input gate, output gate and cell activation vectors respectively, $\boldsymbol{P}_{ix}$ as the weight matrix of the input vector destined to the input gate, $\boldsymbol{P}_{hx}$ as the weight matrix of the hidden layer destined to the input gate, $\boldsymbol{P}_{zx}$ as the weight matrix of the cell vector destined to the input gate, $\boldsymbol{P}_{iy}$ as the weight matrix of the input vector destined to the forget gate, $\boldsymbol{P}_{hy}$ as the weight matrix of the hidden vector destined to the forget gate, $\boldsymbol{P}_{zy}$ as the weight matrix of the cell vector destined to the forget gate, $\boldsymbol{P}_{iz}$ as the weight matrix of the input vector destined to the cell, $\boldsymbol{P}_{hz}$ as the weight matrix of the hidden vector destined to the cell, $\boldsymbol{P}_{iw}$ as the weight matrix of the input vector destined to the output gate, $\boldsymbol{P}_{hw}$ as the weight matrix of the hidden vector destined to the output gate, and $\boldsymbol{P}_{zw}$ as the weight matrix of the cell vector destined to the output gate, we have the Equations 3, 4, 5, 6 and 7 .

$$x(t) = \sigma(P_{ix}i(t) + P_{hx}h(t-1) + P_{zx}z(t-1) + b_x) \tag{3}$$

$$y(t) = \sigma(P_{iy}i(t) + P_{hy}h(t-1) + P_{zy}z(t-1) + b_y) \tag{4}$$

$$z(t) = y(t)z(t-1) + x(t)\tanh(P_{iz}i(t) + P_{hz}h(t-1) + b_z) \tag{5}$$

$$w(t) = \sigma(P_{iw}i(t) + P_{hw}h(t-1) + P_{zw}z(t) + b_w) \tag{6}$$

$$h(t) = w(t)\tanh(z(t)) \tag{7}$$

The sigmoid activation function is always present in the LSTM unit and is represented by Equation 8, while the hyperbolic activation function is represented by Equation 9 and the logistic function is expressed in Equation 10, where $\boldsymbol{L}$ is the curve's maximum value and $\boldsymbol{k}$ relative to the steepness of the same curve.

$$S(x) = exp(x)/(exp(x) + 1) \tag{8}$$

$$\tanh(x) = (1 + exp(2x))/(1 - exp(-2x)) \tag{9}$$

$$f(x) = L/(1 + exp(-k(x - x_0))) \tag{10}$$

3. Related Work

With a similar approach in terms of features, [14] also performs a *hexdump* of the files, involving a set of four bytes, that is, eight hexadecimals for each term considered. This is done through an 8-gram, a contiguous sequence of eights items, in this case, hexadecimals from a malware file. A broader discussion of n-gram in malware classification can be seen in [17], which demonstrates that a large amount of memory may be required and the results tend to overfitting. In addition, deep learning is not addressed and the largest dataset used is smaller than our created for this work.

A malware detection mechanism is discussed in [4]. It is based on the statistical analysis of opcodes (short for operation code), which specify the operations that will be performed at the machine language level. The final results demonstrate that less frequent opcodes are the most important to distinguish different types of malware and cleanware. However, the sampling used for the experiments is small and the famous disassembler used, the IDA Pro[2], has a very expensive license. Our work does not reach this level of machine language and when we deal with 32-bit and 64-bit words, the less frequent question did not prove true in experiments.

In 2015, Microsoft launched a malware classification challenge[3]. The challenge was simply to classify the malware according to their respective families, based on the content and characteristics of the malicious files. A discussion of this Microsoft dataset, including the extraction and selection of the best features, was done in [1]. Exclusive features of static analysis were briefly addressed, which helped us to understand our own work a bit more.

A LSTM, gated recurrent unit (GRU) and convolution neural network (CNN) models are proposed in [2] for the binary malware classification problem. A dataset larger than ours is used but it is not open and publicly available. In addition, the authors perform the classification through systems calls produced by a dynamic analysis. In our work we consider only the static analysis. A summary of these analyzes can be seen in the subsection Malware Analysis.

[2] https://www.hex-rays.com/products/ida/index.shtml
[3] https://www.kaggle.com/c/malware-classification/

The multiclass approach can also be considered through malware families. A malware family consists of malicious codes that share similar features and same behaviors [8]. The complex process of clustering a set of malware into a family usually occurs through API calls and dynamic analysis. This is proposed in [15] which uses a dataset from VirusShare as well but with 51 families of malware labeled among 17.900 samples, five different machine learning algorithms and does not consider the inclusion of cleanwares in the classification.

Several obstacles that try to cause the classifiers of malware have their results worse are inserted in [10]. However, this loss is greater in binary scenarios, and the evaluated dataset is DREBIN that contains malware only from the Android system.

4. Dataset

The dataset created and made available specifically for this work was named MC-dataset-multiclass (abbreviation for malware and cleanware in multiclass scenario). As its name implies, it is made up of a balanced set of malwares and cleanwares, with malwares coming from VirusShare and cleanwares extracted by the authors of this work from different files in Windows systems. We did not find any labeled dataset publicly available with the same amount of data.

We follow the approach proposed by [14], in which the *hexdump* of the malwares is used to generate text files containing the bytes of the original files, represented by hexadecimal pairs. We discard information such as memory address or checksum that are sometimes present. Once this is done, the hexadecimals are converted to numbers that serve as input to our LSTM model. These numbers are read in sets of sixteen hexadecimal digits, which corresponds to a pair of 32-bit words or a 64-bit word.

The MC-dataset-multiclass was created in a totally balanced way in relation to the number of samples for the five classes of malware and one of cleanware, including 3290 samples of each class. We know that the real scenario is composed by more cleanwares than malwares, but we did not want to create any bias for the classification of our model and cleanwares were more difficult to obtain. The labeling of malware classes occurred through VirusTotal[4], which is a website that checks for malwares in user's uploaded files or by md5 hash function, for example. Once this is done, it returns the scan of several antivirus for each file. So what we did was the insertion of the md5 of all samples taken from VirusShare into VirusTotal. This was done by creating json files with all the information returned by VirusTotal for each malware in our dataset.

We count the results of all antivirus for each malware and classified them according to the class returned as the majority. For example, we could imagine the malware file named *malware-1edb420700aa68aba62122b69b805853* classified as trojan by 30 antivirus scanners and 20 classified as backdoor. In this case, we consider malware as belonging to the trojan class. In the case of a tie, we simply did not put the malware in the dataset. All files were named this way with the word *cleanware* or *malware* attached to the respective md5 of each file. Although we know of the existence of "hybrid malwares", we do not consider these cases that would be more related as a multilabel problem.

We also tried a different approach regarding the file types present in the dataset. Most malware analysis articles usually deal with Windows executables only. However, we found that malwares obtained from VirusShare contained different media types (also known as MIME types). This is not entirely reliable because malwares can use a variety of techniques that "trick" this simple identification based on headers and some other parameters. Steganography [12] and encryption [16] are examples of techniques that can easily fool this check. Anyway, we ended up inserting files with similar MIME types in a balanced way but not totally as the number of samples in the dataset. So we tried to "force" our LSTM model to learn these differences a bit, which really happened but we do not know how much exactly. However, Windows executable files are still the vast majority of files in our dataset, both for cleanwares and malwares.

It is important to note that the files selected for the dataset, both cleanwares and malwares, do not exceed something around 3.2 MB. We did this for a better fit in LSTM and also to maintain a reasonable size for uploading the dataset itself. The *hexdumps* were not included in the dataset for size reasons and also because they were an approach created

[4] https://www.virustotal.com/gui/home/upload

exclusively for our work. Another characteristic that needs to be mentioned in our dataset are the 32-bit and 64-bit files. It is not an easy task to check if a file is 32-bit or 64-bit in a set of files, considering the malwares extracted from VirusShare, for example. We tried to put half of the cleanwares samples as 32-bit files and the other half as 64-bit but we know that this amount was not perfectly exact.

The reason for choosing the five malware classes in our dataset was the amount available and extracted from VirusShare. Some other types of malware did not have as many samples following our requirements and we wanted a more balanced dataset. The extraction of cleanwares samples from different Windows systems were not an easy task, we had to install programs from a variety of sources like The Portable Freeware Collection[5], for example. Either way, the five chosen classes can already be classified by threat level, helping in a risk analysis, for example[6]. Finishing, the dataset[7] is now publicly available for download.

5. Model Implementation

One of the concerns with the LSTM was the cost-benefit regarding the time and configuration of the machine for the experiment. We can say that our model is relatively simple and has been tested a lot of times until reaching the desired final version. Our model consists sequentially in an input layer, followed by a LSTM layer, a dropout layer and the final layer, the dense layer. The Figure 2 represents our model implemented in Keras[8], a high-level neural network API that works with the open-source machine learning framework called TensorFlow[9].

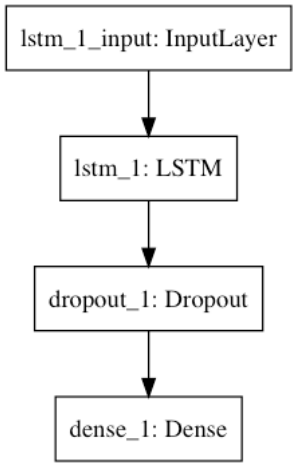

Fig. 2. The LSTM model design implemented in Keras

The input layer sequentially receives the hexadecimal samples of cleanwares and malwares converted to numerical values and passes it on to the LSTM layer. Our model works with a stateful LSTM, which helps in searching for dependencies between the text sequences extracted from the *hexdump* of the dataset files. The LSTM layer has about 128 neurons that are responsible for receiving the inputs of the previous layer and produce outputs using a linear activation function. The number of neurons, time steps and features for the network were chosen after several tests. Time steps and features were suitable to fit the LSTM's reading of at least sixteen hexadecimals at a time, which represents a 64-bit or two 32-bit word in an attempt to not lose information if the sequences were relative to a 32-bit file.

[5] https://www.portablefreeware.com
[6] https://www.kaspersky.com/resource-center/threats/malware-classifications
[7] https://figshare.com/articles/MC-dataset-multiclass/5995468
[8] https://keras.io
[9] https://www.tensorflow.org

Next, we have the "dropout layer" that is actually a technique to reduce overfitting in neural networks [19]. The default dropout value is usually 50% and was the same used in our model. The activation function used in the dense layer was the softmax, described in Equation 11 and usually used when classes are considered mutually exclusive, like in our dataset.

$$f(x_i) = exp(x_i) / \sum_{j=0}^{k} exp(x_j) \qquad \text{for i = 0, 1, 2, ..., k} \tag{11}$$

So we can say that our neural network made use of hexadecimals from the *hexdump* of each file in the dataset. However, we did not read the complete cleanwares and malwares but rather only the most frequent 64-bit or two 32-bit words presents in each file, maintaining the original sequences and using the stateful LSTM. The use of less frequent words has returned worse results, even when we try to address them in conjunction with the more frequent ones. Adaptive moment estimation (Adam) [13] was used as the optimization algorithm with the default values of the parameters present in Keras, but with a small learning rate of 0.0001. The tests were conducted at a computer with a 2.2 GHz Intel Core i7 processor and 8 GB 1600 MHz DDR3 of RAM using only CPU.

6. Experimental Results

The results described in this section are relative to the experiment in the MC-dataset-multiclass. For this multiclass scenario, we used a mini-batch of 14. The amount of backdoors, cleanwares, rootkits, trojans, viruses and worms samples present in the multiclass dataset is 19740. The randomly split of these samples was done as follows: 11844 samples for the training set and 3948 samples for the test set and validation set. We prefered to represent in graphs how the classification of each class was with the classes that we chose for the dataset, as explained in Section 4.

Conversion process using our computer for *hexdump* and Unix *time* command, registered 2304.171 seconds of elapsed real time for all samples of MC-dataset-multiclass. Considering the NumPy array (n-dimensional array object) with numerical values corresponding to the hexadecimals that serve as input to the LSTM, including all the same samples, it took about 52889.327 seconds. The estimate time of arrival (ETA) that is equivalent to one round of training with all the samples, was on average 1141.761 seconds per epoch. We got the best validation accuracy value at epoch 28 after many tests. Test set took about 94.582 seconds to return the final classification results.

True positive rate (TPR) that is an outcome where the model correctly predicts the positive results, can be seen in Figure 3 and the best result 92.19% was with the rootkit class. The worst result was with the trojan class with 51.06%, remembering that the classification did not consider that malwares can have more than one type and trojans are usually the malwares with more subtypes.

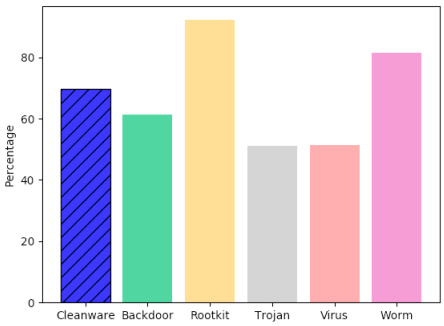

Fig. 3. True positive rate for the six classes

Considering Figure 4, the rootkit class has the best false positive rate (FPR) that is an outcome where our model improperly identifies as cleanware data which is in fact malware, for instance. The rootkit class has a FPR of 3.07% followed by the worm class with 3.93%. The trojan class in an even more discrepant way got the worst result, 11.53%. Again, the rootkit class has the best result of false negative rate (FNR) that is an outcome where the model do an incorrect identification of the results (classifying as malware data which is in fact cleanware, for instance), with 7.81% in Figure 5 and trojan class got the worst, 48.94%. Summarizing, our model has achieved good classification results for some classes, such as rootkit and worm, and bad results for others, such as viruses and trojans. The accuracy was 67.60%. Compressing the classes into two classes, cleanware and malware, we have an accuracy of 90.63%, TPR of 92.76%, followed by a FPR of 8.16% and FNR of 7.24%.

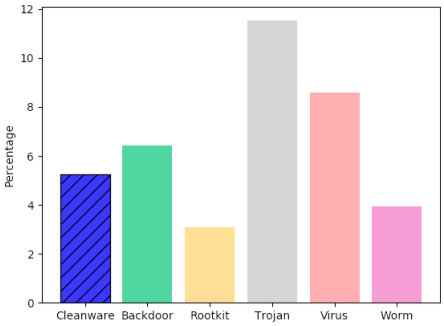

Fig. 4. False positive rate for the six classes

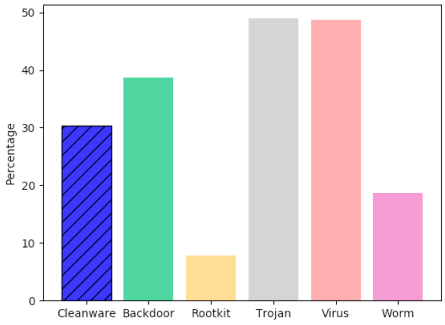

Fig. 5. False negative rate for the six classes

7. Conclusion and Future Work

When we started our work, we knew that it was not an easy task to classify malware using neural networks. In addition, one of our goals was to produce a dataset and make it publicly available for other researches. This has been successfully made but the size of the dataset is still smaller than in some other works. We also try to "escape" somewhat from Windows executable files because of some samples of malware in the dataset, inserting a bit of "noise" and forcing our LSTM model to try to learn these details.

Another concern was the cost-benefit because we observed excellent results in some works but with a very high resource use. We believe that by using fewer resources and maintaining good results it is possible to expand the use of deep learning to more researches and even startups that may arise. Finally, we intend as future work the improvement of the classification results, a multilabel approach and increase the dataset.

References

[1] Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., Giacinto, G., 2016. Novel feature extraction, selection and fusion for effective malware family classification, in: Proceedings of the sixth ACM conference on data and application security and privacy, ACM. pp. 183–194.

[2] Athiwaratkun, B., Stokes, J.W., 2017. Malware classification with lstm and gru language models and a character-level cnn, in: Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on, IEEE. pp. 2482–2486.

[3] Azzouni, A., Pujolle, G., 2017. A long short-term memory recurrent neural network framework for network traffic matrix prediction. arXiv preprint arXiv:1705.05690 .

[4] Bilar, D., 2007. Opcodes as predictor for malware. International Journal of Electronic Security and Digital Forensics 1, 156–168.

[5] Choi, Y.H., Han, B.J., Bae, B.C., Oh, H.G., Sohn, K.W., 2012. Toward extracting malware features for classification using static and dynamic analysis, in: Computing and Networking Technology (ICCNT), 2012 8th International Conference on, IEEE. pp. 126–129.

[6] Egele, M., Scholte, T., Kirda, E., Kruegel, C., 2012. A survey on automated dynamic malware-analysis techniques and tools. ACM computing surveys (CSUR) 44, 6.

[7] Gardiner, J., Nagaraja, S., 2016. On the security of machine learning in malware c&c detection: A survey. ACM Computing Surveys (CSUR) 49, 59.

[8] Gennari, J., French, D., 2011. Defining malware families based on analyst insights, in: Technologies for Homeland Security (HST), 2011 IEEE International Conference on, IEEE. pp. 396–401.

[9] Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y., 2016. Deep learning. volume 1. MIT press Cambridge.

[10] Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P., 2017. Adversarial examples for malware detection, in: European Symposium on Research in Computer Security, Springer. pp. 62–79.

[11] Hochreiter, S., Schmidhuber, J., 1997. Lstm can solve hard long time lag problems, in: Advances in neural information processing systems, pp. 473–479.

[12] Johnson, N.F., Jajodia, S., 1998. Exploring steganography: Seeing the unseen. Computer 31.

[13] Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

[14] Kolter, J.Z., Maloof, M.A., 2004. Learning to detect malicious executables in the wild, in: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM. pp. 470–478.

[15] Pektaş, A., Acarman, T., 2017. Classification of malware families based on runtime behaviors. Journal of Information Security and Applications 37, 91–100.

[16] Rad, B.B., Masrom, M., Ibrahim, S., 2012. Camouflage in malware: from encryption to metamorphism. International Journal of Computer Science and Network Security 12, 74–83.

[17] Raff, E., Zak, R., Cox, R., Sylvester, J., Yacci, P., Ward, R., Tracy, A., McLean, M., Nicholas, C., 2018. An investigation of byte n-gram features for malware classification. Journal of Computer Virology and Hacking Techniques 14, 1–20.

[18] Sen, S., Aydogan, E., Aysan, A.I., 2018. Coevolution of mobile malware and anti-malware. IEEE Transactions on Information Forensics and Security 13, 2563–2574.

[19] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15, 1929–1958.

[20] Sundermeyer, M., Schlüter, R., Ney, H., 2012. Lstm neural networks for language modeling, in: Thirteenth annual conference of the international speech communication association.

[21] Yadav, H., Gour, S., 2014. Cyber attacks: An impact on economy to an organization. International Journal of Information & Computation Technology , 937–940.

[22] You, I., Yim, K., 2010. Malware obfuscation techniques: A brief survey, in: Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on, IEEE. pp. 297–300.