International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 6s
DOI: https://doi.org/10.17762/ijritcc.v11i6s.6894
***Article Received**: 30 March 2023 **Revised**: 05 May 2023 **Accepted**: 25 May 2023*

Detection of Malware in Large Networks using Deep Auto Encoders

K. Janani[1], R. Gunasundari [2]
[1]Research Scholar,Department of Computer Science
Karpagam Academy of Higher Education
Coimbatore, India
jananikumar6@gmail.com
[2]Professor & Head,Department of Computer Applications
Karpagam Academy of Higher Education
Coimbatore, India
*g*unasoundar04@gmail.com

Abstract --- Data mining and machine learning have been heavily studied in recent years with the purpose of detecting sophisticated malware. The majority of these approaches rely on architectures that do not involve deeply enough into the learning process, despite the fact that they have yielded excellent results. This is because deep learning is finding increasing application in both business and academia thanks due to its skills in feature learning. In this paper, we develop a Deep Auto Encoder (DAE) based detection mechanism to detect the malwares crawling in the large scale networks. The DAE act as an unsupervised deep learning model that helps in detecting the malwares. The simulation is conducted on two different datasets to test the robustness of the model. The results show that the proposed method has higher rate of accuracy in detecting the attacks than other methods.

Keywords: feature learning, Deep Auto Encoder, large scale networks, malwares.

I. INTRODUCTION

Due to the pervasive nature of computers and the Internet, it is crucial that sensitive data be safeguarded online. Viruses, worms, trojans, backdoors, spyware, and botnets are all examples of malicious software [1] that are designed to further an attacker illicit goals. Online thieves rely on this method as their primary method of attack for a wide variety of security breaches, putting users at risk of catastrophic injury and financial loss [2].

Internet Security Threat Report (ISTR) data shows that malware infections led to the loss or theft of 500 million personal records [3] and that up to $1 billion was stolen from financial institutions throughout the world in just over two years [4]. Consequently, both the anti-malware market and academics place a premium on effective malware identification [5].

Anti-malware software is the first and last line of defence against malicious programs. There was an early emphasis on signature-based detection techniques [6]. Signature is a strategy developed by researchers with the intention of accurately classifying future instances of known malware while minimizing the number of false positives. Each piece of malicious software has its own short string of bytes, or signature. The virus is classified according to this string. However, malware attackers can easily circumvent this procedure by using techniques like as encryption, polymorphism, and obfuscation [7].

Most anti-malware software addresses this challenge by keeping tabs on what malicious code is up to in the system kernel. This prevents harmful software from modifying user data. Compared to static detection, dynamic detection is more robust, but it also has a higher initial cost and does not scale well. As a result of financial incentives, hundreds of malicious files are created and distributed daily [8], making it challenging for detection systems to be effective. These systems make use of a variety of methods, including data mining and machine learning.

Models for identifying malware are constructed in these systems with the help of classification techniques [9]. Most of these approaches are grounded in relatively simple instructional frameworks. Even though shallow learning architectures showed some promise when used to virus detection, they ultimately fell short. This evolution in malware writing techniques has resulted in an ever-increasing influx of newly discovered file samples that require constant analysis. This is essential work that has to be completed [10].

However, these unlabeled files are rarely discarded during the machine learning process because they always reflect the trend of malware development and fresh releases of

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 6s
DOI: https://doi.org/10.17762/ijritcc.v11i6s.6894
***Article Received**: 30 March 2023 **Revised**: 05 May 2023 **Accepted**: 25 May 2023*

innocuous apps. This is because sensitive data may be hiding in plain sight in files that have not been properly named. Therefore, there is a lot of potential for development in this sector [11].

Deep learning is at the forefront of machine learning and is already being used in several fields [12-15]. The optimal architecture for feature learning incorporates several layers of deep learning and gives the system access to both labelled and unlabelled data samples. When building a deep learning architecture, it is common practice to train multiple layers of feature detectors from scratch before building the final classification model [16]. This allows the architecture to overcome the learning difficulties. Due to this, we decided to develop a malware detection system based on deep learning. In this paper, we develop a Deep Auto Encoder (DAE) is used detect the malwares in a large-scale network.

1. Background

Kolosnjaji et al. [17] developed one-hot encoding to transform the API request sequence into a series of binary vectors. Machine learning benefits from one-hot encoding. This model scores very highly in accuracy (89.4%) and precision (85.6%). Its recall rate of 89.4% is also very high.

The purpose of the malware detector developed by Tobiyama et al. [18] was to extract features from an API call log over time. Then, a convolutional neural network (CNN) evaluates the visual representation of these attributes to decide whether or not the object in question should be classified as malicious. While the RNN achieves its results with the aid of a long short-term memory (LSTM), the CNN makes use of four convolutional layers and four pooling layers to do the same thing. After that, a link is established between the two next levels. They managed to get an AUC of 0.96 despite dealing with a rather small dataset.

For the purpose of extracting n-grams, Ding et al. [19] made use of the operational codes. Those who used the DBN were not privy to all three of its layers. There are a total of 10,000 files in the dataset; 3,000 were safe, 3,000 were harmful, and 3,000 were not categorized in any way. When compared to other DBNs, the top performer achieved a 96.7% success rate in accuracy.

McLaughlin et al. [20] successfully developed a detector using the opcodes present in malware files without resorting to feature selection or engineering. A CNN that was used to process the raw opcode data. An embedding layer came first, and then each of the other layers. Precision was between 99% and 27%, recall was between 95% and 85%, and F1 scores varied from 97% to 78%, depending on the dataset.

When working with software binaries, Saxe and Berlin [21] transformed them into 2D entropy histograms [22]. The software did not necessitate any form of filtering, unpacking, or categorizing on the user part to accomplish this. Common DNN was trained with these features as inputs so that it could classify. Incorporating these characteristics allows you to train a neural network with four layers, a sigmoid activation function.

The likelihood that a given file includes malicious code was subsequently evaluated using a Bayesian calibration model developed by Saxe and Berlin [21]. Since it is not reasonable to assume that the classifier has a normal distribution, we utilize a prior on the ratio of dangerous software to benign software and the error rate of the deep neural network (DNN) to estimate the kernel density. They claim a 95% detection rate and a false positive rate (FPR) of 0.1%; both of these numbers are within acceptable ranges.

II. PROPOSED METHOD

In this section, we use unsupervised autoencoders for feature extraction and classification of features in an unsupervised manner. In case of large-scale networks, the classification is carried out based on the network logs to classify the malwares.

An autoencoder takes a vector as input, and the goal of the network is to produce an output that is a perfect representation of the vector. Both directions will work for this purpose. These neural networks are extremely flexible because of their capacity to learn unsupervised compression encoding. They can also be trained incrementally, layer by layer, which drastically reduces the computer resources needed to produce an accurate model.

The network encoding example shown in Figure 3 is one such network. The hidden layers of this network have fewer dimensions than the exposed ones at the input and output levels. Denoising autoencoders are trained to reconstruct the original input from a noisy one by removing noise from the original input. Due to this, they are more reliable than standard autoencoders. This method has been shown to be more flexible and dependable than conventional autoencoders. Some have seen parallels between the two.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 6s
DOI: https://doi.org/10.17762/ijritcc.v11i6s.6894
***Article Received**: 30 March 2023 **Revised**: 05 May 2023 **Accepted**: 25 May 2023*

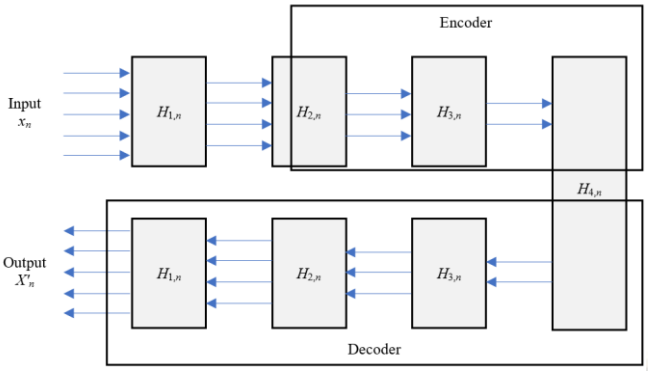

Figure 2: Deep Auto encoder

1.1. Deep Auto Encoders

Figure 2 illustrates that an auto-encoder is a neural network model that functions by making an effort to produce outputs that are identical to the inputs it gets. This ensures that the model predictions are as accurate as possible. The data that is fed into a basic auto-encoder and the data that is output from the auto-encoder both encounter some kind of change during the process of encoding and decoding. This is because encoding and decoding are both processes that involve transformation of data. The data that is being input is converted into encoded data by the encoder component. The encoded data can then be used. The data that has been encoded is then taken by the decoder component and used to recreate the data that is being created. Auto-encoders are useful tools that allow users to properly replicate input data using reduced-dimensional features in a way that is both efficient and quick.

An auto-encoder has a two constituent parts, which are known respectively as an encoder and a decoder. The Figure 1 makes it clear that the variable *x* stands for the data source, *y* for the encoded data, and *x*' for the decoded data. This information may be found by looking at the graphic. In order to carry out their respective mapping processes, the encoder function (g) and the decoder function (h) both make the assumption that the gap between the data that was input and the data that was output is suitably narrow.

It is possible to use the backpropagation method in an unsupervised learning scenario with a neural network if the desired outputs of an auto-encoder neural network with equal inputs. Setting the desired outputs of an auto-encoder neural network to be equal to the network inputs. As can be seen in Figure 1, an auto-encoder is able to be disassembled into its primary components, which can be considered its building blocks. There are three main components that make up the structure: the input, the hidden, and the output layer. In order to find a representation of the inputs that is shared by the networks, AE are employed extensively in DNN. Encoder and decoder are the two distinct components of the system that may be distinguished from one another.

The model only contains a single input layer, a single hidden layer, and a single output layer. There are a total of eight nodes in the network, with four located in the input layer, two located in the hidden layer, and four located in the output layer. It is essential to remember that the +1 nodes represent the bias of each individual node in the network.

Encoder

In the event that an input of the form $x \in R^n$ is provided, the value $h(x)$ will be used to represent the hidden layer (x).

$$h(x)=f(W^{(1)}x+b^{(1)})$$

where

$f(\cdot)$ - activation function.

The study uses logistic sigmoid function as the activation function and it is represented as:

$$f(x)=1/(1+\exp(-x))$$

Decoder

The reconstructed value x' corresponds exactly to the representation $h(x)$ of the topmost layer, which is.

$$X'=f(W^{(2)}h(x)+b^{(2)})$$

where

$W(2) \in R^{n \times m}$ - output weight matrix,

$b(2) \in R^n$ - output bias vector.

The reconstruction error, denoted by the symbol D(x,x'), is defined as follows for any given set x of input data:

$$D(x,x')=\sum \|x-x'\|^2.$$

A model of auto-encoding that is capable of deep learning and has the capability to do so. The notation h1, h2, etc., hm1, and hm represent the characteristics that are recovered by each auto-encoder. hm stands for high-order mode. The raw data is what is fed into the first DAE while the extracted feature by the AE came before it is what is fed into the auto-encoders that follow it in succession.

The network is typically trained using the batch gradient descent method, which is an approach that we have encountered in the past and which helps limit the number of mistakes that can occur during reconstruction.

If the dimension of the input is more than the size of the hidden layers, then this is the result. The name spatial auto-encoders is the one that is most commonly used to refer to

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 6s
DOI: https://doi.org/10.17762/ijritcc.v11i6s.6894
***Article Received**: 30 March 2023 **Revised**: 05 May 2023 **Accepted**: 25 May 2023*

this specific category of AE. Following that, the cost function can be finished as follows:

$$L(W,b) = 0.5\lambda\sum\|W^{(l)}\|^2+\beta KL(\rho\|\rho') + 2n^{-1}\|h_{W,b}(x)-y\|^2$$

where

third term - sparsity regularization term and

β - sparsity penalty parameter.

$KL(\rho\|\rho')$ - relative entropy

ρ - sparsity parameter and

ρ' - mean hidden node activation using parameters *b* and *W*.

Deep auto-encoders

Once upon a time, it was thought that neural networks consisting of only three layers were examples of shallow learning networks. The levels in question were input, hidden, and output respectively. The information that possesses obvious input characteristics lends itself particularly well to the operation of these networks. When dealing with intricate input data, however, it is reasonable to predict the requirement of a network that contains more hidden layers. Hidden layers in neural networks can, in practice, be understood as nonlinear changes of the layers that come before them. Because of this, the network is able to learn a more intricate structure based on the data that it is provided with. As a direct result of this, it is possible for us to end up constructing a deep auto-encoder network by training a very large number of auto-encoders in conjunction with one another.

As can be seen in Figure 4, each consecutive auto-encoder uses as its input the features that were previously extracted by the auto-encoder that came before it. This is because each succeeding auto-encoder builds on the work of the auto-encoder that came before it. Another name for this kind of network is stacked auto-encoders, but it also goes by a few other names as well.

DAE, on the other hand, have the capability of recognizing characteristics that shallow structures are unable to recognize. Their research is focused mostly on the following two ideas:

- *Pre-training*: At this point, an unsupervised greedy training strategy is being utilized to train the deep neural network one layer at a time, independently. It is vital to bear in mind that each layer is trained individually, and the representation that was acquired by the layer that came before it is utilized as the input for the layer that comes after it. This is something that should be kept in mind at all times. To obtain a set of weight parameter values that are relatively near to what would be optimal, pre-training is a method that can be utilized.
- *Fine-tuning*: At this point, the complete deep network will be trained using supervised methods. The training will take place. The unprocessed data serves as the input for the deep neural network, which then generates, as its output, a representation that is based on what it has learned at the very highest level. After then, classification problems might be solved using this representation. There is a possibility that the information that was obtained from the input will also be found in the output. As the restored data corresponds to our anticipated outcome, we decided to use the reconstruction error as the assessment indicator for this particular research.

III. RESULTS AND DISCUSSIONS

In a binary classification competition, a model success can be measured in a variety of ways. There is often more than one name for a particular metric. Four values from the confusion matrix, a comparison of the computed predicted class to the ground truth, will serve as the basis for all of the assessment metrics that will be presented here.

Accuracy:

One way to evaluate how well an analysis performed is by calculating the proportion correct, often known as accuracy. Accuracy has less of an impact when there is a disparity between socioeconomic groups.

$$acc=(TP+TN)/(TP+TN+FP+FN)$$

Sensitivity:

It is the fraction of items properly identified as belonging to class x relative to the total items in a class x, which is defined as the sensitivity, true positive rate, chance of detection, or recall.

$$TPR=(TP)/(TP+FN).$$

Specificity:

The proportion of objects that could be confidently categorized as not X is frequently referred to as the True Negative Rate (TNR),

$$TNR=(TN)/(TN+FP).$$

F1 Score (F1):

The F1 Score is the harmonic mean of the accuracy (p) and the false-positive rate (r).

$$F1=2(p*r)/(p+r).$$

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 6s
DOI: https://doi.org/10.17762/ijritcc.v11i6s.6894
***Article Received**: 30 March 2023 **Revised**: 05 May 2023 **Accepted**: 25 May 2023*

In this specific scenario, the F-function was designed to give top priority to the genuine positive rate.

To test our theory, we looked at the CTU-13 and MalRec datasets, which contain information about malware-induced network behavior. Two separate experiments were conducted. The next step of the study will be to utilize the most successful malware datasets to develop a method for detecting network activity.

Table 1: Dataset Specifications

Dataset	Parameter	Value
Malrec	Malware Recorded	66,301
	Hashing	MD5
	Network Activity	PCAP form
CTU-13	Total Recordings	13 captures or scenarios

Our investigation made use of the C# and Python programming languages, as well as the Scikit-Learn module for Python. To conduct our research, we used a server that boasted 512 GB of RAM and 32 CPU cores.

Several publicly available datasets were used during the preliminary testing phase. These included MalRec and CTU-13. All MalRec reports, including the one responsible for the traffic spike, have been copied and added to the dataset. AVClass [14] and other malware labeling tools, we can organize samples in accordance with the families to which they belong. We tallied up how many samples each household sent in and ranked them from most to least prolific in order to find out which 25 families had sent in the most total samples.

In the second set, there are 24,197 malware samples representing many different malware families. To keep it from being further classified, it is decided to give it its own distinct category. Since our objective is to discover the top five types of dangerous software, we have opted to analyze the most frequent malware families individually. This list was compiled through the analysis of botnet traffic and is based on the CTU-13 dataset.

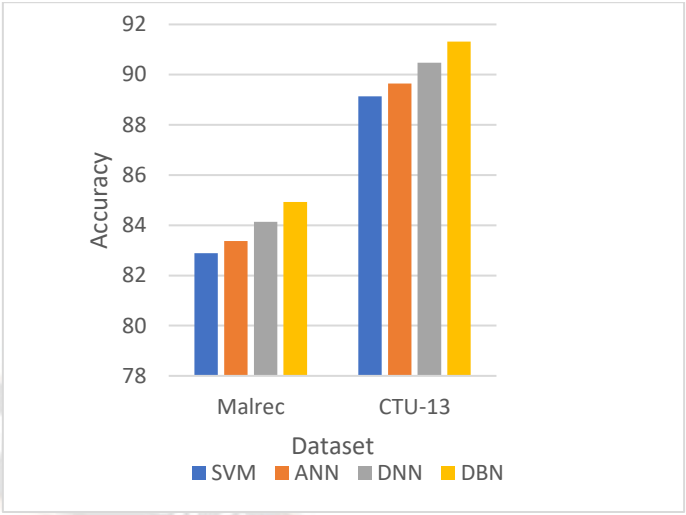

Figure 2: Accuracy

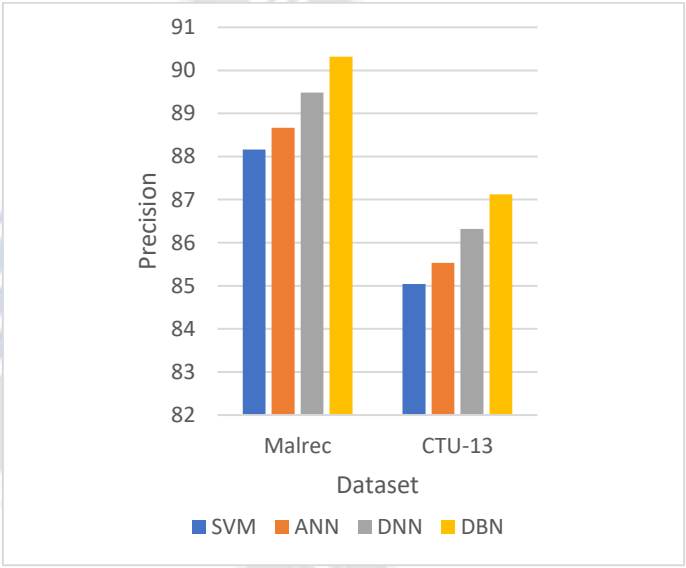

Figure 3: Precision

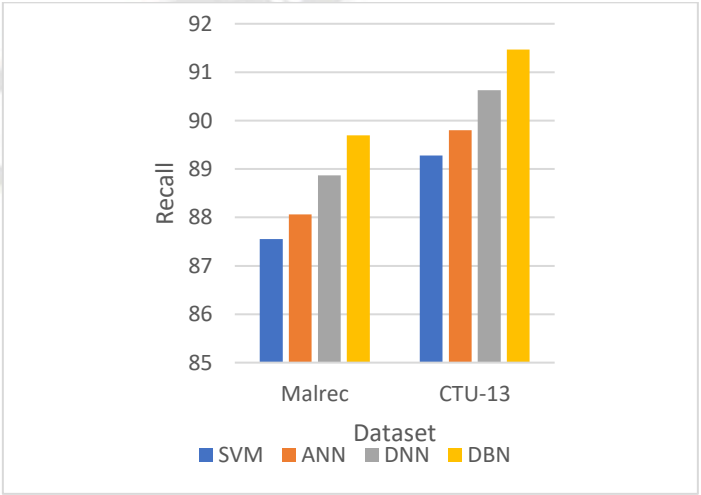

Figure 4: Recall

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 6s
DOI: https://doi.org/10.17762/ijritcc.v11i6s.6894
Article Received: 30 March 2023 Revised: 05 May 2023 Accepted: 25 May 2023

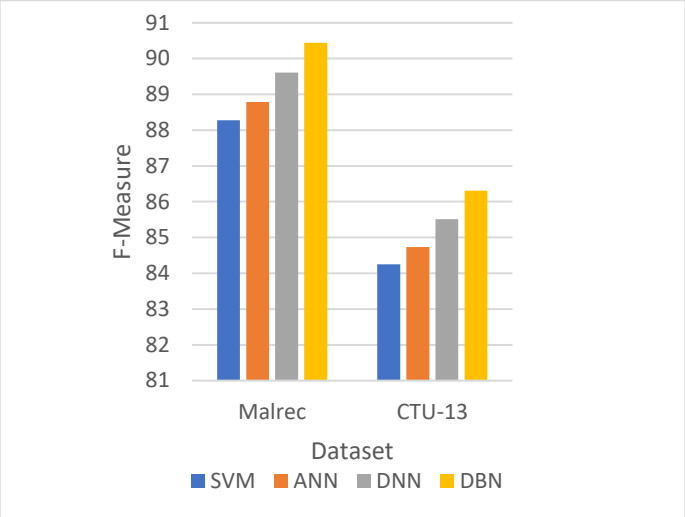

Figure 5: F-Measure

From the results it is seen that the proposed method has higher rate of accuracy, precision, recall and f-measure as in Figure 2-5.

IV. CONCLUSIONS

To identify malicious programs in massive networks, we create a detection approach based on deep auto encoders. The use of deep auto encoders as an unsupervised deep learning model can aid in the detection of malicious programs. To ensure the consistency of the model, the simulation is run on two independent data sets. The results show that the proposed strategy outperforms competing methods in its ability to reliably detect attacks.

REFERENCES

[1] Abusitta, A., Li, M. Q., & Fung, B. C. (2021). Malware classification and composition analysis: A survey of recent developments. *Journal of Information Security and Applications*, *59*, 102828.

[2] Aslan, Ö., & Yilmaz, A. A. (2021). A new malware classification framework based on deep learning algorithms. *Ieee Access*, *9*, 87936-87951.

[3] Awan, M. J., Masood, O. A., Mohammed, M. A., Yasin, A., Zain, A. M., Damaševičius, R., & Abdulkareem, K. H. (2021). Image-Based Malware Classification Using VGG19 Network and Spatial Convolutional Attention. *Electronics*, *10*(19), 2444.

[4] Gibert, D., Planes, J., Mateu, C., & Le, Q. (2022). Fusing feature engineering and deep learning: A case study for malware classification. *Expert Systems with Applications*, *207*, 117957.

[5] Kumar, S. (2021). MCFT-CNN: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in Internet of Things. Future Generation Computer Systems, 125, 334-351.

[6] Dib, M., Torabi, S., Bou-Harb, E., & Assi, C. (2021). A multi-dimensional deep learning framework for iot malware classification and family attribution. *IEEE Transactions on Network and Service Management*, *18*(2), 1165-1177.

[7] Xiao, M., Guo, C., Shen, G., Cui, Y., & Jiang, C. (2021). Image-based malware classification using section distribution information. *Computers & Security*, *110*, 102420.

[8] Yadav, B., & Tokekar, S. (2021). Recent innovations and comparison of deep learning techniques in malware classification: a review. *International Journal of Information Security Science*, *9*(4), 230-247.

[9] D'Angelo, G., Ficco, M., & Palmieri, F. (2021). Association rule-based malware classification using common subsequences of API calls. *Applied Soft Computing*, *105*, 107234.

[10] Mallik, A., Khetarpal, A., & Kumar, S. (2022). ConRec: malware classification using convolutional recurrence. *Journal of Computer Virology and Hacking Techniques*, 1-17.

[11] Lu, Q., Zhang, H., Kinawi, H., & Niu, D. (2022). Self-Attentive Models for Real-Time Malware Classification. *IEEE Access*.

[12] Yoo, S., Kim, S., Kim, S., & Kang, B. B. (2021). AI-HydRa: Advanced hybrid approach using random forest and deep learning for malware classification. *Information Sciences*, *546*, 420-435.

[13] Rizvi, S. K. J., Aslam, W., Shahzad, M., Saleem, S., & Fraz, M. M. (2022). PROUD-MAL: static analysis-based progressive framework for deep unsupervised malware classification of windows portable executable. *Complex & Intelligent Systems*, *8*(1), 673-685.

[14] Elkabbash, E. T., Mostafa, R. R., & Barakat, S. I. (2021). Android malware classification based on random vector functional link and artificial Jellyfish Search optimizer. *PloS one*, *16*(11), e0260232.

[15] Kale, A. S., Pandya, V., Di Troia, F., & Stamp, M. (2022). Malware classification with Word2Vec, HMM2Vec, BERT, and ELMo. *Journal of Computer Virology and Hacking Techniques*, 1-16.

[16] Kolosnjaji, B., Zarras, A., Webster, G., & Eckert, C. (2016, December). Deep learning for classification of malware system call sequences. In *Australasian joint conference on artificial intelligence* (pp. 137-149). Springer, Cham.

[17] Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T., & Yagi, T. (2016, June). Malware detection with deep neural network using process behavior. In *2016 IEEE 40th annual computer software and applications conference (COMPSAC)* (Vol. 2, pp. 577-582). IEEE.

[18] Ding, Y., Chen, S., & Xu, J. (2016, July). Application of deep belief networks for opcode based malware detection. In *2016 International Joint Conference on Neural Networks (IJCNN)* (pp. 3901-3908). IEEE.

[19] McLaughlin, N., Martinez del Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., ... & Joon Ahn, G. (2017, March). Deep android malware detection.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 6s
DOI: https://doi.org/10.17762/ijritcc.v11i6s.6894
Article Received: 30 March 2023 Revised: 05 May 2023 Accepted: 25 May 2023

In *Proceedings of the seventh ACM on conference on data and application security and privacy* (pp. 301-308).

[20] Saxe, J., & Berlin, K. (2015, October). Deep neural network based malware detection using two dimensional binary program features. In *2015 10th international conference on malicious and unwanted software (MALWARE)* (pp. 11-20). IEEE.

[21] Weber, M., Schmid, M., Schatz, M., & Geyer, D. (2002, December). A toolkit for detecting and analyzing malicious software. In *18th Annual Computer Security Applications Conference, 2002. Proceedings.* (pp. 423-431). IEEE.