JOURNAL OF SCIENCE
OF LAC HONG UNIVERSITY
ISSN: 2525 - 2186
Tạp chí Khoa học Lạc Hồng, **2025**, 20, 06-12

PHÁT HIỆN MÃ ĐỘC TRONG FILE PE SỬ DỤNG HỌC SÂU VỚI KỸ THUẬT HỌC TỰ GIÁM SÁT

Võ Khương Lĩnh[1*], Nguyễn Hòa Nhật Quang[2]
[1]Trường Đại học Nguyễn Huệ, phường Tam Phước, thành phố Biên Hòa, tỉnh Đồng Nai, Việt Nam
[2]Trường Đại học Trần Đại Nghĩa, số 189, đường Nguyễn Oanh, phường 10, quận Gò Vấp, TP. Hồ Chí Minh, Việt Nam
** Tác giả liên hệ: vokhuonglinh@gmail.com*

THÔNG TIN BÀI BÁO

Ngày nhận: 20/01/2025
Ngày hoàn thiện: 24/2/2025
Ngày chấp nhận: 7/3/2025
Ngày đăng: 15/3/2025

TỪ KHÓA

Biểu diễn mã độc;
Phát hiện mã độc;
Học sâu;
Mạng nơ-ron tích chập;
Học tự giám sát.

TÓM TẮT

Trong những năm gần đây, đã có sự gia tăng đột biến về phần mềm độc hại mới do tin tặc tạo ra trên toàn cầu, đặt ra thách thức cho các phương pháp phát hiện truyền thống. Bài báo này khám phá việc sử dụng trí tuệ nhân tạo tiên tiến, cụ thể là Học sâu với Học tự giám sát, để xác định phần mềm độc hại trong các tệp thực thi. Nghiên cứu của chúng tôi tập trung vào việc so sánh hiệu quả của các kỹ thuật học sâu phổ biến như mô hình CNN và mô hình CNN tinh chỉnh, với các mô hình Autoencoder. Đóng góp chính của bài báo này nằm ở việc so sánh kết quả của các phương pháp tiếp cận khác nhau này để phát hiện phần mềm độc hại.

MALWARE DETECTION IN PE FILES USING DEEP LEARNING WITH SELF-SUPERVISED LEARNING TECHNIQUES

Vo Khuong Linh[1*], Nguyen Hoa Nhat Quang[2]
[1]Nguyen Hue University, Tam Phuoc Ward, Bien Hoa City, Dong Nai Province, Vietnam
[2]Tran Dai Nghia University, No. 189, Nguyen Oanh Street, Ward 10, Go Vap District, Ho Chi Minh City, Vietnam
*Corresponding Author: *vokhuonglinh@gmail.com*

ARTICLE INFO

Received: Jan 20th, 2025
Revised: Feb 24th, 2025
Accepted: Mar 7th, 2025
Published: Mar 15th, 2025

KEYWORDS

Malware representation;
Malware detection;
Deep learning;
Convolutional neural network;
Self-supervised learning.

ABSTRACT

In recent years, there has been a surge in new malware created by hackers globally, posing challenges for traditional detection methods. This paper explores using advanced artificial intelligence, specifically Deep Learning with Self-supervised learning, to identify malware in executable files. Our study focuses on comparing the effectiveness of popular deep learning techniques like CNN models and fine-tuned CNN models, against Autoencoder models. The key contribution of this paper lies in comparing the results of these different approaches to malware detection.

Doi: https://doi.org/10.61591/jslhu.20.608
Available online at: https://js.lhu.edu.vn/index.php/lachong

1. INTRODUCTION

In recent times, with the development of artificial intelligence, the explosion in the number of malware strains, as well as their variations, is one of the challenges that cause certain difficulties for traditional malware detection methods in the field of Information Security. Therefore, automatic malware detection is essential. Malware detection based on deep learning is one of the methods that brings positive results and is suitable for current requirements.

Today, deep learning has been applied in many fields, especially with good results in image recognition. Due to the many hidden layers between the input and output layers, deep learning models can extract features and classify data into defined classes. When a file is represented as an image, a trained deep learning model can determine or predict whether it is malware.

Portable executable files (PEs) have an important role in information security as they are typically containers for malware and execute malicious behaviors. This type of file contains executing machine codes used to start programs and applications on the computer. It is an important part of the system and one of the main sources of malware deployment.

There are many approaches to using Deep Learning to detect malicious code, such as using Recurrent networks [14], Neural Networks [6, 11, 24], and Convolutional Neural Networks [2, 4, 11, 16, 17]. According to research, this paper has found that the use of Convolutional Neural Networks to detect malicious code (with their image representation) yields the good results. Furthermore, some studies promote solving the problem of detecting malware in their image representation using unsupervised learning models. [7, 18, 25, 26]

Within the scope of the article, this research focuses on self-supervised learning model for malware images detection problem. This paper compares and evaluates the experiment's result with other learning methods, using models that have been studied.

2. RELATED WORK

2.1 Convolutional Neural Network

There are many Deep Learning techniques for solving the malware detection problem. However, the studies [12, 15, 17] proved that the Convolutional Neural Network yields the best results.

A Convolutional neural network (CNN) is a deep learning architecture designed to process and analyse visual data, such as images and videos by automatically learning features between pixels [6]. The main components of CNN include convolutional layers, pooling layers, fully connected (dense) layers, activation functions, and an overfitting reduction method through dropout [13]. In particular, convolutional layers help detect and extract features and characteristics of images (edges, textures, patterns…), while pooling layers play the role of reducing the complexity of the computing process [4].

The approach using deep learning in malware detection has yielded many good results due to several advantageous features that CNNs offer:

- Feature Extraction and Learning: CNNs are adept at automatically learning hierarchical representations of data and relevant features from raw data, eliminating the need for manual feature engineering.

- Spatial Hierarchical Learning: CNNs can learn spatial hierarchies of features within the data, which is useful for detecting complex patterns in the structure and content of malware.

- Scalability: CNNs can be scaled to handle large datasets, which is critical in the context of malware detection due to the vast number of malware samples that need to be processed.

By leveraging these capabilities, researchers can develop robust and effective malware detection systems capable of identifying both known and unknown malware variants. The models proposed in [19] and [5] have structures and parameters shown in Table 1 and Table 2, respectively. Through experiments, this research found that in the model [19], executing one step requires 100ms, while model [5] requires approximately 900ms in each epoch.

Most studies, such as [5, 9, 19], use networks with a structure of three interleaved convolutional layers and three pooling layers. Within the scope of this study, the paper will also apply and compare the two models of [5] and [19] on two datasets: EMBER dataset and a self-built dataset to obtain an overall objective view.

2.2 Image Representation of Malware

Representing malware as images is a technique used in some cases, especially when leveraging CNNs for malware detection.

To implement CNNs in malware detection, it is necessary to represent malware as images. According to the proposal by L. Nataraj and colleagues [9], both benign and malware files are considered binary strings composed of 0s and 1s. Then, those strings can be read as a byte sequence (8 bits) arranged in a two-dimensional array and represented as a grayscale image where each pixel has a value ranging from 0 to 255. Variants of the same malware strain exhibit similar image representations regarding pixel pattern features, while different malware strains will have distinct images, differing from benign files (Figure 1).

The challenge is to determine the optimal image representation method to preserve or enhance the distinctive characteristics of the image. Through observation, there are two main factors that lead to the loss of feature information: pixel arrangement order and the process of determining the size of the original image.

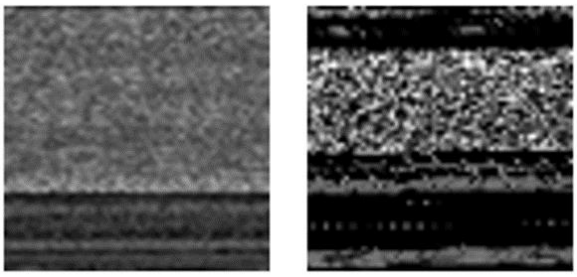

Figure 1. *Grayscale Image Representation of Benigh (left) and Malware (Right).*

In most research articles on using CNNs in the problem of detecting malicious code, they often use the method of arranging pixels in a regular order. For example, L. Nataraj [9] and Minh Tu et al. [19] used the this method. The regular order for pixel arrangement is the allocation of pixels from left to right and top to bottom of the image.

They established the thresholds depending on the byte sequence length to determine the width size of the image. The width size will be from 32, 64, 128, 256, 354, 512 and 1024. Then, the height will be determined by dividing the sequence length by the width. Most of the time, the output images are usually rectangular in shape (the height is usually longer than the width). The images were then resized to 64x64 dimensions.

In other studies for grayscale images, Quach Danh Ngoc [5] and Ren Z. [15] proposed new approaches that mitigate one limitation of [9, 19]. They used different methods of pixel arrangement such as Gray, Zigzag, H-curve, Hilbert curve, Z-order, and Sweep curve. Only Hilbert and Zigzag methods keep the distance between any two consecutive points on the byte string placed next to each other in 2-D space. after the transformation into an image.

This is general malware analysis and detection. Particularly with PE files, the characteristic information that helps distinguish malware from benign is mostly concentrated in the header, which is the beginning part of the file's byte sequence. Thus, Moreina et al. [12] and [10] used a diagonal zigzag patterns to detect ransomware while transforming PE header data into images.

Therefore, with the zigzag order of pixel arrangement method, the header information of the PE file will be extended horizontally within some top rows of the image. And the Hilbert curve method has difficulty converting to custom sized images. Furthermore, with the rectangle size determining process, the operation of resizing the image to 64x64 (square) size will increase the risk of information distortion.

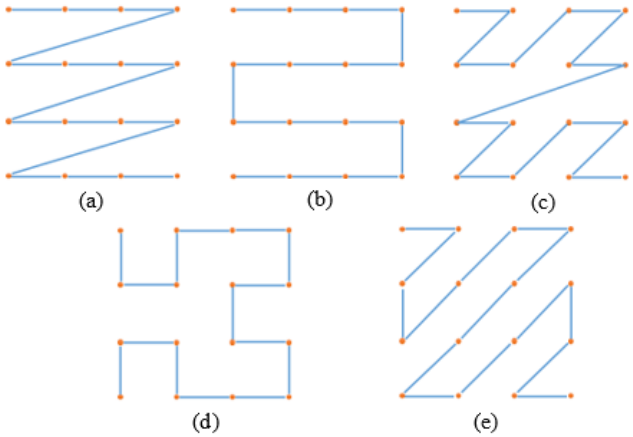

Figure 2. *Image representation patterns regular (a), zigzag (b), z-order (c), hilbert-curve (d) and serpentine (e).*

Through studying the above research, this study chose the zigzag pixel arrangement method based on the image's diagonal called "serpentine" order for experiments. This method gave good results, in particular, diagonal zigzag patterns are suitable for PE files as well as preserving the continuity between pixels.

2.3 Self-supervised learning technique

There is a study [17] using self-supervised learning for malware detection with a natural language processing approach, which gave better results than supervised learning. On the other hand, there are research [3, 7] using Self-Supervised Learning for anomaly detection which also gave better results than supervised learning technique. Therefore, this paper proposes an approach of using Self-Supervised Learning technique to serve malware detection problem.

In AI field, Self-Supervised Learning (SSL) is a paradigm where a model is trained on a task and, instead of depending on external labels provided by humans, uses the data itself to generate supervisory signals. Self-supervised learning in neural networks seeks to generate meaningful training signals by taking advantage of innate structures or relationships in the input data. The way SSL tasks are made, completing them necessitates identifying key characteristics or connections in the data. Usually, the input data is enhanced or changed in a way that produces related sample pairs. The supervisory signal is created using one sample as the input and the other as the output. Noise addition, cropping, rotation, and other adjustments are examples of this augmentation. Self-supervised learning more closely resembles how humans classify objects in the real world.

An artificial neural network or another model, like a decision list, is the foundation of the conventional SSL technique. There are two steps in the learning process for the model. Initially, the task is completed by applying pseudo-labels to help initialize the model parameters, based on an auxiliary or pretext classification task. Second, either supervised or unsupervised learning is used to complete the task at hand. Completing patterns from masked input patterns is the focus of other auxiliary tasks.

In computer vision and natural language processing (NLP), where the quantity of labeled data needed to train models can be unreasonably large, self-supervised

learning is especially helpful. Self-supervised model training is more economical and time-efficient because it requires fewer annotations and labels on the data.

Therefore, in the scope of this study, with the SSL idea, the image representation of malware and benign files will be augmented to leverage inherent structures and relationships within the input data to create meaningful training signals.

3. PROPASAL FOR PE FORMAT MALWARE DETECTION USING AUTOENCODER

As mentioned in the previous section, this study proposes an approach to use Self-Supervised Learning models for malware detecting with image representation. First, the input files will be converted into grayscale images with serpentine pixel arrangement methods. After that, those original images will be augmented transforming (flipping, rotating, jigsaw puzzling). Then, the augmented images are going into an Autoencoder to be reconstructed into original image. Thus, this process helps getting the core patterns aka the most distinctive features to classify if they are malware or benign.

Autoencoder belongs to unsupervised deep learning network group. It gains knowledge of effective data representation or encoding. Since it is a directed neural network that is less dependent on training data and is capable of producing its own labeled data, it is unsupervised deep learning - more precisely, self-supervised deep learning. The purpose of Autoencoder is to try to reproduce the input data as closely as possible. Autoencoder is often used in data dimensionality reduction problems, image noise removal or anomaly detection. So, this article focuses on the problem of malware detection.

An Autoencoder network can be divided into three main components: encoder f(x), code (also known as “bottleneck”) h and decoder g(h). The bottleneck or code layer is the representative layer, usually the smallest size in the network, the main function of this layer is to store the most important information from the input data. Meanwhile, the encoder layer tries to put the input data into the code layer, and the decoder layer tries to recreate the output data from the code layer. If we consider x as the input data and r as the data reproduced from the decoder layer, we can understand: h = f(x) and r = g(h). Autoencoder is essentially a neural network, so it can be trained through back-propagation with an error function.

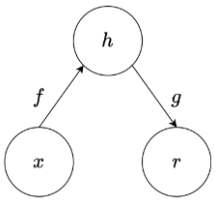

Figure 3. *Basic structure of Autoencoder.*

The encoder consists of a series of convolutional blocks, pooling modules, and a bottleneck, which is a compact area where the model's input is compressed. The decoder, which consists of several upsampling modules to reconstruct the compressed feature as an image, comes after the bottleneck. When using a basic autoencoder, the desired result should be the same as the input data but with less noise. On the other hand, variational autoencoders create an entirely new image using the input data that the model has been given.

The bottleneck is the smallest and most crucial component of the neural network. The purpose of the bottleneck is to limit the amount of data that can pass from the encoder to the decoder, allowing only the most essential information to do so. We can say that the bottleneck aids in the formation of a knowledge-representation of the input since it is constructed in a way that captures the maximum amount of information contained in an image. As a result, the encoder-decoder structure aids in obtaining the maximum amount of data from an image and creates valuable associations between different network inputs. The neural network is further prevented from memorizing the input and overfitting on the data by a bottleneck acting as a compressed representation of the input. The smaller the bottleneck, the lower the risk of overfitting, but very small bottlenecks would limit the amount of data that could be stored, which raises the possibility that crucial data could escape the encoder's pooling layers.

Eventually, the output of the bottleneck is reconstructed by a series of convolutional and upsampling blocks that make up the decoder. The decoder functions as a "decompressor," reconstructing the image from its latent attributes, since the input is a compressed knowledge representation.

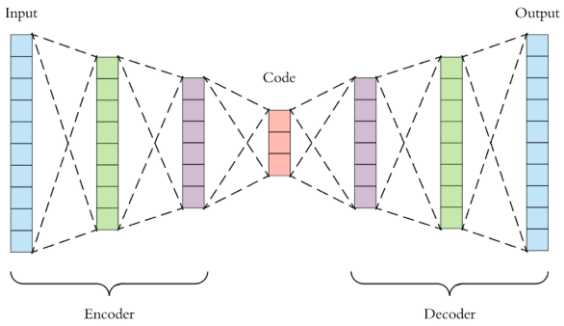

Figure 4. *Deep Autoencoder structure: Encoder, Bottleneck (code), Decoder.*

To reproduce the input data as closely as possible to the training data is the aim of the autoencoder. With this knowledge, we can provide Autoencoder with only data of the benign class - not malicious code - for it to learn. Next, we determine the reconstruction error for both malware and benign components of the dataset. Reconstruction error indicates whether the Autoencoder's reconstruction of the benign set is accurate or not. A smaller reconstruction error indicates a higher reconstruction error. This indicates that the input data is malicious software rather than benign data. The reconstruction error now resembles a histogram; all we need to do is determine a threshold to distinguish between the malicious and benign error sets. This turns the issue into a binary classification issue.

4. EXPERIMENTS AND EVALUATION

4.1 Dataset Description

This study uses 2 datasets, one is EMBER [8] (with data from 2017), and the other is self-built called LQDU-23 (with data from 2023). The EMBER dataset contains total 1,100,000 samples with the distribution shown in figure 5 below:

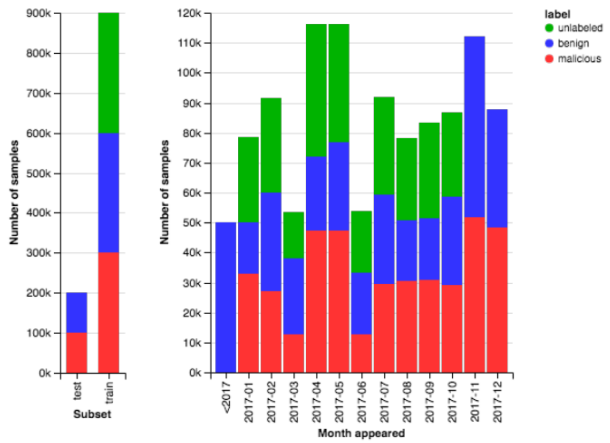

Figure 5. *Distribution of samples in EMBER dataset.*

The EMBER dataset [8], collected in 2017, has many types and has been balanced in terms of the number of files of each type. Types of malicious code in EMBER include: adware, backdoor, mining, flooder, packed, ransomware, riskware, rootkit, spyware, common trojan, virus, worm, etc. Ensure the number of files for each is about 40,000 files.

On the other hand, this study builds a dataset called LQDU-23, and benign files are collected, all of which are files with the PE format or with PE file extensions belonging to the PE group. Malware files are collected from the VirusShare [20] channel, which came from the VirusShare_00468.zip set, with a total size of 52.63 GB and containing 27533 malware files in PE format. They were uploaded on April 30, 2023. These samples are then uploaded to VirusTotal [21] to scan and retrieve the results to classify malware, as well as serve as a basis for separating the training and testing sets. After that, training and test sets are built with 2000 samples of benign and 2000 samples of malware in each set.

In the LQDU-23 dataset, the number of strains collected is not much, with most focusing on trojan strains (including droppers, downloaders, ransomware, and proxies); a few are viruses and worms; this is also reflected in the report. Actual reports in CIS [23]:

Table 1. *Distribution of malware families in LQDU-23*

Type	Quantity
Trojan	16193
Virus	1177
Worm	343
Others	273

Our approach is to experimentally apply the methods to a multi-species, symmetric, balanced (EMBER) data set and then use the same training results to test on another dataset , which is unbalanced in terms of type (LQDU-23) to test the performance quality of the methods against each other: specifically, the ability to extract features hidden within the structure of the malicious code. Especially in real-life situations, the type of malware used tends to change depending on the characteristics of the technological situation, the security capabilities of software developersand the level of security awareness of users' networks.

Since this paper uses 4 different methods to transform the original images to become the input for Autoencoder, the data size will be increased by 4 times. Thus, in the scope of this study, 2000 samples were randomly chosen from each subset with a fixed seed parameter.

After obtaining the dataset for training and testing, this study proceeds to convert them into grayscale images according to the method of L. Nataraj et al. [9]: consider the data bytes as a pixel, arrange them into photos in order from top to bottom, left to right. Then resize the image to 64x64 to fit the input of a neural network.

From here, create two copies of the above image dataset. A set will be labeled as malicious or benign. This dataset is ready for two methods. CNN [18] and fine-tuned CNN [5]. The remaining unlabeled copy of the dataset will be augmented into 4 versions: rotated 90 degrees, rotated 270 degrees, flipped horizontally, and flipped vertically, as shown in Figure 6:

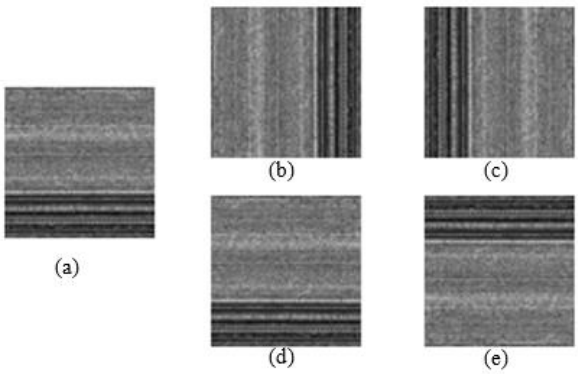

Figure 6. *Augmented Images: (a) original, (b) rotate 90 degree, (c) rotate 270 degree, (d) flip-horizontal, (e) flip-vertical*

4.2 Experimental results & evaluation

The experiment program is built in Python 3 language with the Keras library, running on a computer with the following configuration:

- Intel Core i7-6700HQ CPU @2.60GHz,

- RAM: 12GB RAM,

- NVIDIA GeForce GTX 960M graphics card.

The experiment uses 3 learning models: two CNNs from [5, 19] and SSL with Autoencoder to train with EMBER dataset. After that, the trained models are tested on both the EMBER and LQDU-23 datasets.

Due to the equal number of malware and benign in the training dataset, the "accuracy" measurement is used. In

addition, this study also uses "recall" and "f-score" measurements.

For the CNN method, training and generating results follow the model outlined in [5, 19]. Particularly with the Autoencoder method, this study uses the trained encoder from the autoencoder to compress input data and train a different predictive model. After that, a logistic regression model is processed to train on the training dataset directly and evaluate the performance of the model on the holdout test set, as shown in Figure 7.

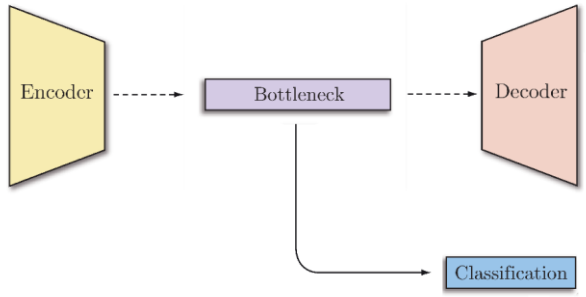

Figure 7. *Using Autoencoder for classification*

The experimental results, with the EMBER dataset, the difference of all three models is not too different, while with the LQDU-23 dataset, Autoencoder method still maintains high accuracy while the other two models decrease sharply as shown in Table 2.

Table 2. *Experiment Results*

Dataset / Approach	EMBER			LQDU-23		
	Acc	Recall	F1	Acc	Recall	F1
CNN [19]	92.06	92.06	93.94	83.03	83.03	90.73
CNN [5]	96.33	96.33	97.95	92.22	92.22	95.95
Autoencoder	96.31	96.31	97.57	94.73	94.73	97.09

This shows that with Autoencoder, underlying features are extracted and are more effective in detecting different variations of malicious code. From there, it can be used to detect malicious code in new data sets, shorten update times, and retrain the classifier. According to experimental results, this method shows high development potential, meeting the requirements in today's malware situation.

The results of our study suggest that when it comes to spotting different types of malware, not all methods are equal. This study trained three models - CNN, fine-tuned CNN, and Autoencoder using the EMBER dataset, which is a solid collection of malware from 2017. As above, all three models did pretty well on this dataset.

However, when this research tested these models on the LQDU-23 dataset, which has newer malware up until 2023, the Autoencoder model stood out. It kept its high accuracy, unlike the CNN and fine-tuned CNN models, which struggled more with the new kinds of malware.

This tells us that the Autoencoder method is good at picking up on the hidden, underlying features of malware. It's better at recognizing new types of malicious code. This could mean faster updates for security systems and easier retraining of detection tools.

5. CONCLUSION

In summary, the primary contribution of this paper is the comparison of three distinct Deep Learning methods: CNN, fine-tuned CNN, and Autoencoder for malware detection in executable files. This article tested these techniques on LQDU-23, a more recent dataset, as well as EMBER, a reliable dataset with older malware. Our experiment shows that the Autoencoder model was the most effective at identifying novel and evolving malware types. Even with the most recent threats from up to 2023, it remained accurate. This implies that the Autoencoder technique is potential for enhancing the training process of detection tools and shorten security system updates time and resources. The Autoencoder's adaptability is primarily due to its capacity to extract underlying patterns from malware images. The self-supervised learning model can offer a promising approach to detect and classify malware generated by AI. This work adds knowledge to the discussion of self-supervised learning applications in practical problem solving, especially malware detection.

6. REFERENCES

[1] Anh Tran Ngoc, Linh Vo Khuong. Malware detection based on Machine Learning and PE header information. *Information Security Journal* **2021**. Vietnam.

[2] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. International Conference on Neural Information Processing Systems (NIPS) **2012**.

[3] Chun-Liang Li, Kihyuk Sohn, Jinsung Yoon, Tomas Pfister. CutPaste: Self-Supervised Learning for Anomaly Detection and Localization, Computer Vision Foundation **2021**.

[4] Gibert D. Convolutional neural networks for malware classification. University Rovira i Virgili, Tarragona, Spain, **2016**.

[5] Hung Nguyen Viet, Ngoc Quach Danh, Dung Pham Ngoc. Research on techniques of representing malware files and deep learning models in malware detection, XXII National Conference: Some selected issues of Information and Communication Technology **2019**. Thai Binh, Vietnam.

[6] Kephart J.O. Tesauro, G.J., Gregory B Sorkin. Neural networks for computer virus recognition. *IEEE International Conference on Intelligence and Security Informatics* **1996**.

[7] Hadi Hojjati, Thi Kieu Khanh Ho, Naregs Armanfard, Self-Supervised Anomaly Detection: A Survey and Outlook **2023**. Montreal, QC, Canada,

[8] Hyrum S. Aderson, Phil Roth. EMBER: An open dataset for training static PE malware machine learning models, arXivLabs, Cornell University **2018**.

[9] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. Malware images: Visualization and automatic classification, Proceedings of the 8th International Symposium on Visualization for Cyber Security **2011**.

[10] Linh V. K., Hung Ng. V., Anh Tr. Ng. Enhance Deep Learning model for malware detection with a new image representation method, *Information Security Journal* **2023**., Vietnam.

[11] Li Deng George E. Dahl, Jack W. Stokes and Dong Yu. Large-scale malware classification using random projections and neural network **2013**. ICASSP.

[12] Moreira, C. C., Moreira, D. C., & de Sales Jr, C. D. S. Improving ransomware detection based on portable executable header using xception convolutional neural network, Computers & Security **2023**. 130, 103265.

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting J. Mach. Learn **2013**. Res. 15(1):1929–1958.

[14] Razvan Pascanu, Jack W. Stokes, Li Deng, Dong Yu, Mady Marinescu, Anil Thomas. Malware Classification with Recurrent Networks **2015**. IEEE ICASSP.

[15] Ren, Z., Chen, G., & Lu, W. Malware visualization methods based on deep convolution neural networks, Multimedia Tools and Applications **2020**. 79, 10975-10993.

[16] Sunoh Choi, Sungwook Jang, Youngsoo Kim, Jonghyun Kim. Malware Detection using Malware Image and Deep Learning. International Conference on Information and Communication Technology Convergence **2017**, Jeju, Korea (South).

[17] Seonhee Seok, Howon Kim. Visualized Malware Classification Based on Convolutional Network. Journal of The Korea Institute of Information Security and Cryptology **2016**.

[18] Setia Juli Irzal Ismail, Hafiz Pradana Gemilang, Budi Rahardjo, Hendrawan. Self-Supervised Learning Implementation for Malware Detection. International Conference on Wireless and Telematics (ICWT) **2022**.

[19] Tu Nguyen Minh, Hung Nguyen Viet, Anh Phan Viet, Loi Cao Van, Nathan Shone. Detecting Malware Based on Dynamic Analysis Techniques Using Deep Graph Learning. Lecture Notes in Computer Science **2020**, vol. 12466.

[20] VirusShare – free malware storage, https://virusshare.com/. Accessed: 2023-05-01.

[21] Virustotal – free online malware scanner, https://www.virustotal.com/. Accessed: 2023-04-30.

[22] Website Center for Internet Security CIS - A community-driven nonprofit, responsible for the CIS Controls and CIS Benchmarks, https://www.cisecurity.org/. Accessed: 2023-08-23

[23] Wenyi Huang, Jack W. Stokes, MtNet: A Multi-Task Neural Network for Dynamic Malware Classification, DIMVA, **2016**.

[24] Xiaofei Xing, Xiang Jin, Haroon Elahi, Hai Jiang, Guojun Wang, A Malware Detection Approach Using Autoencoder in Deep Learning, IEEE Access **2022**. Vol. 10.

[25] Xin Li, Peixin Lu, Lianting Hu, XiaoGuang Wang, Long Lu. A novel self-learning semi-supervised deep learning network to detect fake news on social media. Multimedia Tools and Applications **2022.**