


Article

Automatic Malicious Code Classification System through Static Analysis Using Machine Learning

Sungjoong Kim ¹, Seongkyu Yeom ¹, Haengrok Oh ², Dongil Shin ¹ and Dongkyoo Shin ^{1,*} 

¹ Department of Computer Engineering, Sejong University, Seoul 05006, Korea; tjdwnd2004@sejong.ac.kr (S.K.); dae02159@naver.com (S.Y.); dshin@sejong.ac.kr (D.S.)

² Agency for Defense Development, Seoul 05661, Korea; haengrok@add.re.kr

* Correspondence: shindk@sejong.ac.kr

Abstract: The development of information and communication technology (ICT) is making daily life more convenient by allowing access to information at anytime and anywhere and by improving the efficiency of organizations. Unfortunately, malicious code is also proliferating and becoming increasingly complex and sophisticated. In fact, even novices can now easily create it using hacking tools, which is causing it to increase and spread exponentially. It has become difficult for humans to respond to such a surge. As a result, many studies have pursued methods to automatically analyze and classify malicious code. There are currently two methods for analyzing it: a dynamic analysis method that executes the program directly and confirms the execution result, and a static analysis method that analyzes the program without executing it. This paper proposes a static analysis automation technique for malicious code that uses machine learning. This classification system was designed by combining a method for classifying malicious code using a portable executable (PE) structure and a method for classifying it using a PE structure. The system has 98.77% accuracy when classifying normal and malicious files. The proposed system can be used to classify various types of malware from PE files to shell code.

Keywords: malicious code; classification; portable executable (PE)



Citation: Kim, S.; Yeom, S.; Oh, H.; Shin, D.; Shin, D. Automatic Malicious Code Classification System through Static Analysis Using Machine Learning. *Symmetry* **2021**, *13*, 35. <https://doi.org/10.3390/sym13010035>

Received: 3 December 2020

Accepted: 24 December 2020

Published: 28 December 2020

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The development of information and communication technology (ICT) is making daily life more convenient by allowing access to information at anytime and anywhere and by improving the efficiency of organizations [1]. However, various cyber attacks such as information leakage and ransomware have also been increasing. Most of these cyber attacks are caused by malicious code.

Malware is becoming increasingly complex and sophisticated as a result of evolving computer technology. When the open source concept emerged, various types were generated, and now even novices can easily create malicious code using hacking tools. Such code is increasing and spreading exponentially [2–4]. Approximately 20% of all malicious code in circulation is classified as variants of existing code [5]. Thus, various types have been developed, further increasing malware attacks and making it difficult for a person to manually respond to such malicious code. As a result, a considerable amount of research has been done to automatically analyze and classify it.

There are currently two methods for analyzing malicious code: a dynamic analysis method that executes the program directly and confirms the execution result, and a static analysis method that analyzes the program without executing it. The dynamic analysis method monitors changes that occur as the file is executed and checks what function it performs. Although it has the advantage of analyzing changes during the actual execution, there is a limit in the ability to analyze all the execution paths [6]. A significant limitation is that this method cannot analyze a malicious code that triggers underlying behavior, such as only working at a specific point in time. Thus, it is difficult to analyze and classify it

effectively using only a dynamic analysis automation system. Therefore, it is necessary to pursue research and systems that utilize technologies that can automate the static analysis of harmful code.

This paper proposes a static classification system for malicious code that combines machine learning and deep learning. Section 2 describes malicious code classification research, and Section 3 proposes a system architecture. Section 4 presents the results of experimental and performance analyses, and Section 5 presents the implications and conclusions related to the system.

2. Related Works

Various studies have been conducted regarding the analysis and classification of malicious code. This includes extracting the application programming interface (API) used in the program and calculating the weight based on the probability that the API will occur in malicious code [7], converting it to the N-gram method, and extracting the entire N-series element binary file into a string to create a general file, which is then applied to malicious code classification [7]. Static levels of packing and obfuscation signs have been used to calculate the level of risk and examine the structure of the binary files to investigate the extent of the malicious code [7].

Nataraj proposed a method to visualize and classify malicious code files, using an image gabor filter to train the classifier as an image feature extractor, along with a k-nearest neighbors (kNN) classifier [8]. Chen suggested DroidVecDeep, which is a malicious code detection method that uses deep learning technology to effectively detect unknown code on the Android platform. DroidVecDeep first extracts various features and ranks them using the mean decrease impurity. Then, it transforms the features into compact vectors based on word2vec. Finally, it trains the classifier based on a deep learning model [9].

Naeem proposed the cross-platform malware variant classification system (CP-MVCS), which converts binary malicious code into a grayscale image and extracts malicious functions from images using combined SIFT-GIST malware (CSGM) [10].

Other studies have classified malicious code based on information extracted from the portable executable (PE) header and section information [11,12].

3. Design and Implementation of Malicious Code Classification System

A malicious code classification system is proposed to automate a static analysis to distinguish and classify the nature of the file itself without running it. The designed classification system receives all of the files as input data, including the malicious code, normal file, and source file. Figure 1 shows the structure of the whole system.

In the preprocessing stage, the PE data extraction module and the image generation module are used to generate input data for each module used in the classification stage. In the next classification step, each model individually judges whether it is malicious using several algorithms. Random forest, gradient boosting, and decision tree algorithms classify malicious codes by receiving data generated from PE data, and CNN algorithm classifies malicious codes by receiving images generated by the image generation module as input data. By integrating the classification results of each model, the final malicious code is determined. Finally, this is the step of reflecting the classification results in the DB. DB configuration consists of data information and a value that determines whether the data is malicious.

Progress in the system is largely divided into a preprocessing step, malicious code classification step, and step to reflect the results in a database (DB). The contents of each step are as follows.

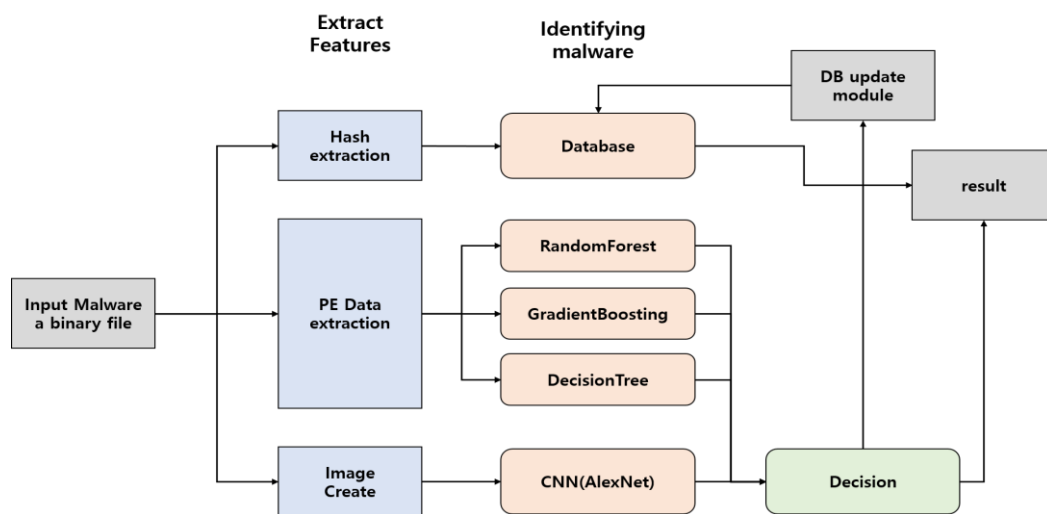


Figure 1. Overall structure of proposed malicious code static classification system.

3.1. Preprocessing Step

The system has a learning model that has been trained with various kinds of algorithms. In order to extract and process the input file to enter data into this model, it extracts hash values from input files, extracts PE data, and performs image conversion work.

3.1.1. Hash Extraction

This step extracts the hash value of an input file, which is an eigenvalue. This is done to determine whether the input data has been duplicated. Using the extracted hash value as a primary key, the classification result of the newly entered data in the DB update step is added to the DB, and the duplicated data is modified in the DB.

3.1.2. PE Data Extraction

The information needed for PE files to run in Windows exists in the header and sections of the PE structure. Therefore, information related to malignancy can be obtained from PE structures without executing the malicious code, and the import address table (IAT) within the PE header can be used to determine which dynamic link library (DLL) is loaded, and which function is used in that DLL [13]. If the data has a PE structure, a total of 55 features, including the entropy and packers, are extracted from the header and section parts of the file. At this time, the YARA rule setting is used to find the packing information of the file in the binary file. The YARA rule is composed of tools to identify and classify malicious code types using their signatures. In a traditional malicious code classification system, if the patterns are compared and judged to be malicious, it is used as a considered in the proposed system. Figure 2 shows an example of the attribute information extracted from the data by the YARA rule.

3.1.3. Image Create

Image Create is a module that visualizes the entered file for the CNN and converts it into image data. The input data are treated as a one-dimensional vector.

Name	Machine	Characteristics	Subsystem	ImageBase	Checksum	MajorLinkerVersion	...
3DVision.exe	332	259	2	4194304	17222466	4.975	...
7-zip.dll	34404	8226	2	268435456	0	4.975	...
7-zip32.dll	332	8494	2	268435456	0	4.975	...
7z.exe	34404	8226	2	268435456	0	4.975	...
7za.exe	34404	35	3	4194304	0	4.975	...
...

Figure 2. Examples of features in PE structure extracted from malicious code.

First, this one-dimensional vector is converted to a two-dimensional vector using the following equation. Equation (1) is an equation that calculates the length of one side of the image and converts it to a 2D vector. Use of Equation (1) allows to find the size and convert it to a 2D vector based on the size found.

$$n = \text{floor}(\sqrt{\text{FileSize}}), \tag{1}$$

Second, the data converted to a 2D vector is used to generate a color image of a 3D vector by assigning a byte value of 0–255 to each ID value for the converted vector using the basic palette of Deluxe Paint. Figure 3 shows the results of converting the file to an 8-bit color image.

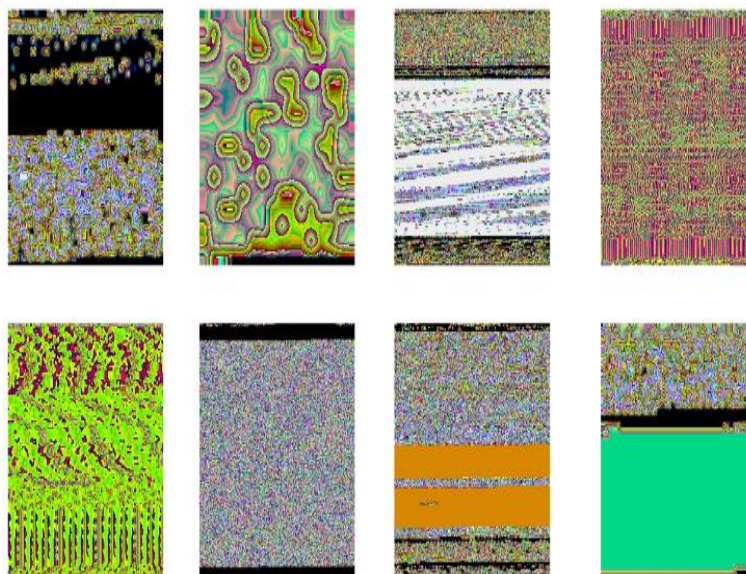


Figure 3. Binary file converted to 8-bit color image.

Finally, to apply it to the CNN, it is necessary to change the size of the generated images to the same size. OpenCV is used to convert the size of a generated image to a 256 × 256 image. Figure 4 shows the pseudo code for the image creation.

Algorithm : Image_Create_module

```

#Step. 1 : Convert for Malware Binary data to image

dataList = input(Binary_file)
dataListSize = floor(sqrt(length(dataList)))
dataList = reshape(dataListSize, dataListSize)
for y in range(0,length(dataList)):
    for x in range(0,length(dataList[y])):
        image[y][x] = dataList[y][x]

#Step. 2 : Resize image

image = resize(image,(224,224))
for y in range(0,length(image)):
    for x in range(0,length(image[y])):
        r,g,b = Deluxe_Paint()
        image[y][x] = [r,g,b]

```

Figure 4. Pseudo code for image creation.

3.2. Classification Step

The malicious code classification proceeds by using the pre-processed data. Various experiments have been done to find a model suitable for classification. Details of these experiments are described in Section 4.

3.2.1. Classification Using PE Structure

For modules that use PE structures, the classification of the malicious code is performed using the decision tree, random forest, and gradient boosting algorithms, which have excellent performances. Rather than using all the properties of the PE structure, an automated feature selection method is used to score the importance and then classify using the feature with the highest importance score. Table 1 lists 12 attributes extracted from 54 attributes.

Table 1. Twelve features extracted out of 54 attributes using automatic feature selection.

	Attribute	Importance Value
1	ImageBase	0.233
2	MajorSubsystemVersion	0.137
3	Machine	0.097
4	DllCharacteristics	0.075
5	Characteristics	0.047
6	Subsystem	0.039
7	MinorSubsystemVersion	0.031
8	LoadConfigurationSize	0.031
9	VersionInformationSize	0.022
10	MajorOperatingSystemVersion	0.021
11	SectionsMaxEntropy	0.020
12	SectionAlignment	0.019

3.2.2. Classification Using Image

The image module uses AlexNet [14] to proceed with the classification. The detailed layers and parameters of the classifier are given in Table 2.

Table 2. Parameters of proposed AlexNet (CNN).

Layer	Step	Parameter	Value	Output
0	Input		FILE	$224 \times 224 \times 3$
1	Conv1	Filter n	32	$74 \times 74 \times 32$
		Filter Size	5×5	
		Filter stride	3	
		Channel	3	
	Max_Pool1	Filter Size	3×3	$37 \times 37 \times 32$
		Filter stride	2	$13 \times 13 \times 28$
2	Conv2	Filter n	128	
		Filter Size	5×5	
		Filter stride	3	
		Channel	32	
	Max_Pool2	Filter Size	3×3	$11 \times 11 \times 128$
		Filter stride	1	
3	Conv3	Filter n	192	$5 \times 5 \times 192$
		Filter Size	5×5	
		Filter stride	2	
		Channel	128	
	Max_Pool3	Filter Size	3×3	$3 \times 3 \times 192$
		Filter stride	1	
4	Max_Conv4	Filter n	256	$2 \times 2 \times 156$
		Filter Size	3×3	
		Filter stride	1	
		Channel	192	
5	FC1			128
6	FC2			2

3.2.3. Final Classification Result

The result is selected based on the maximum frequency among the classification results of the four models, and a determination is finally made about whether it is malicious code. If the frequency is the same, the classification is conducted again. If the result of the reclassification is the same, the result is considered malicious code, because it is more dangerous to view normally if it is considered as malicious code. Figure 5 shows the pseudo code of the decision module. In Figure 5, good and bad indicate the number of normal and malicious codes.

3.3. Database Application Step

This module corrects the weight value in the database as a result of classifying whether or not the code is malicious. When the weight value reaches a specific value in the DB, there is an advantage that the classification speed is increased by distinguishing whether a malicious or non-malicious DB value is seen without passing through the model in the next classification. If the file is not registered with the DB, it is first registered with the DB before starting the operation. Figure 6 shows the operational process of the DB update module.

```

Algorithm : decision module

#Variables
vote : analysis result counting
good : Classification result normal count
bad : Classification result malicious code count

#Step. 1 : Judgment of analysis result
good = vote(result[true])
bad = vote(result[false])

#Step. 2 : Final result
if good > bad :
return good
else
return bad
    
```

Figure 5. Pseudo code for decision module.

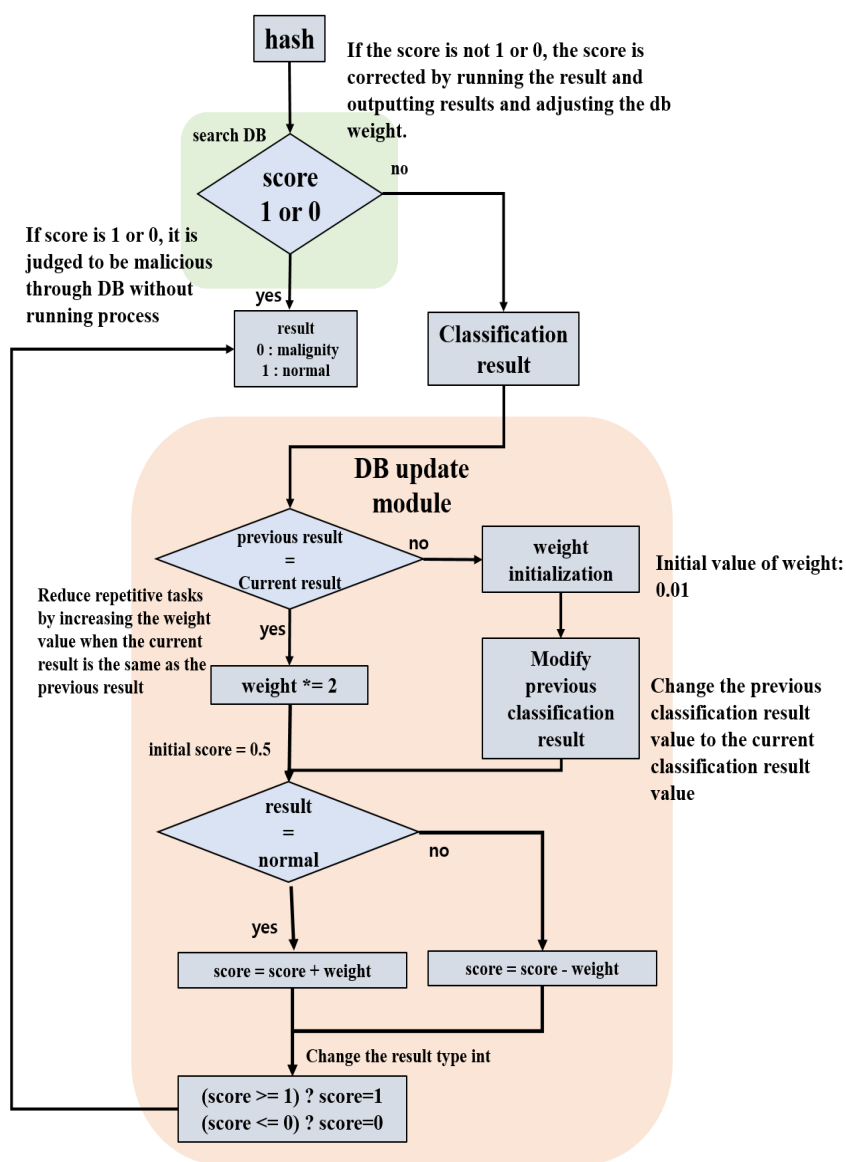


Figure 6. Operation process of DB update module.

4. Experiment

In order to select an algorithm for the classification of malicious code, experiments were performed to classify various malicious code samples for each module. The three experiments listed in the following subsections were performed.

4.1. Module Using PE Structure

The binary classification used random malware from VX Heaven [15]. This included data from a typical Windows portable program with approximately 1000 files extracted from 10,000 files. In order to find an algorithm suitable for the classification of malicious code, experiments were done with five algorithms. The algorithm was carried out using the sklearn [16] library. The first algorithm was AdaBoost, which was repeated 50 times based on a decision tree. The second algorithm was random forest, with no 10 depth limit using bootstrap gradient boosting without prior probability. The third algorithm was a decision tree with the maximum depth limited to 10. The fourth algorithm was logistic regression, and the last algorithm was Gaussian Naive-Bayes(GNB) with 50 booster repeats. In addition, experiments were conducted to classify groups of malicious code. The verification method was learned with 80% of the data, and the cross verification method used the remaining 20% of the data. Figure 7 shows the detailed results of the malicious code classification experiment on a module using PE information. The blue bar is the result accuracy of the binary classification and the orange bar is the result accuracy of the malicious code classification.

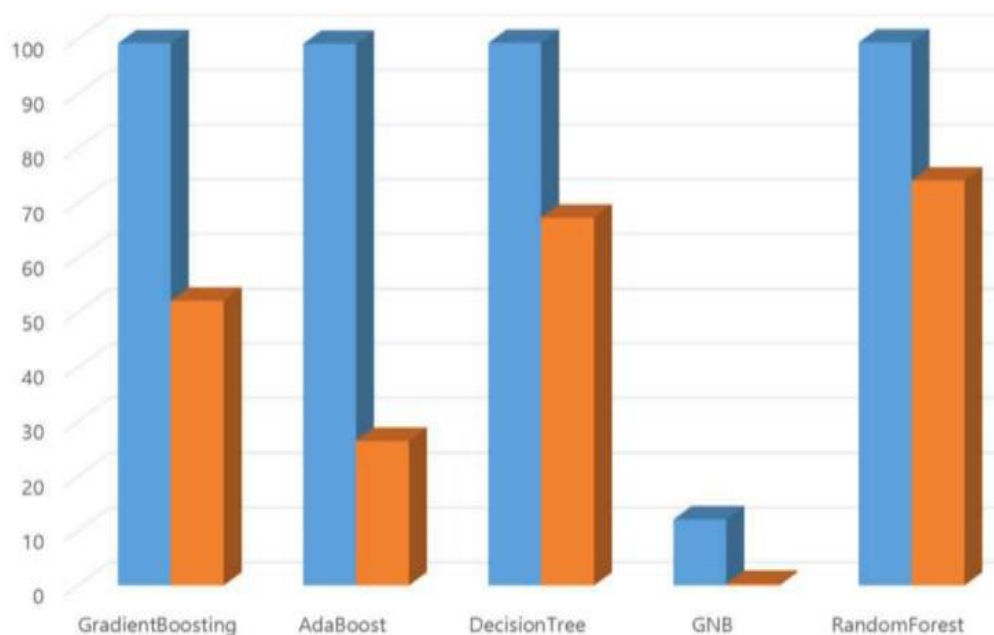


Figure 7. Results of malware classification experiment for modules using PE information.

4.2. Module Using Image

The performance of the classifier was tested using the Microsoft Malware Classification Challenge (Big 2015) dataset [17]. Of the nine existing classes and 10,868 items in the training data, a bytecode file with the PE header removed was used, with 20% used to verify the training and 80% for random split. To check the performance of various pre-treatment methods, an experiment was conducted by organizing the data into black and white, and color images through Deluxe Paint mapping, and then the color images with three bytes were grouped together into a single pixel. The experiment was conducted using TensorFlow [18]. The learning results are shown in Figure 8.

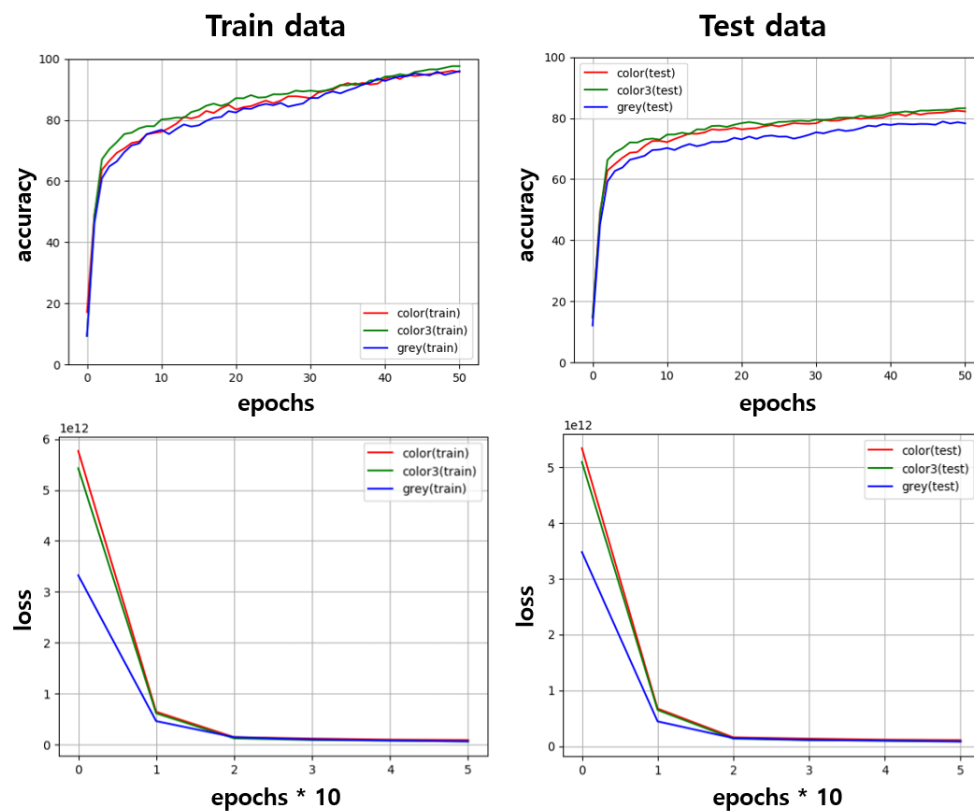


Figure 8. Experimental results for image data.

Three experiments for each data set showed similar accuracy in tests with training data, but re-verification using untrained test data showed that the accuracy with a black and white image was approximately 3% lower than with the two color images. The loss rate showed that black and white images were chosen more appropriately than color images, and the overall learning speed was higher.

4.3. Proposed Classification System Performance Experiment

In order to evaluate the performance of the designed malicious code classification system, normal and malicious file classification experiments were conducted. Malicious files were randomly selected from approximately 23,000 files from VX Heaven data, with the normal files consisting of approximately 1100 DLLs and executables from Windows. The experimental results showed an accuracy of approximately 98.77%. The detailed accuracy results are listed in Table 3. As the learning progressed, the classification time decreased, while maintaining the accuracy of the DB.

Table 3. Performance evaluation of the proposed system.

Experiment	Answer	
	Positive	Negative
Positive	1066	138
Negative	159	22,958

5. Conclusions

As the proliferation of malicious code increases and it becomes ever more intelligent, there is insufficient manpower to analyze all of it and respond manually. To overcome this problem, this paper proposed a system that automatically and statically analyzes the code to determine if it is malicious. Various characteristic factors are extracted, such as the hash

value, PE metadata, and packer information, and classified using various machine learning algorithms. This is similar to the existing automatic signature-based classification, which is an automatic analysis tool, but differs because the system is used as a consideration, not a pattern to judge whether the code is malicious. In addition, the file itself is visualized through a visualization method and entered into a CNN model. Thus, both PE files and shell-like files are classified.

In the future, this method can be improved through experiments and research to classify various types of malicious code information, instead of just determining the existence of malicious code. In addition, using the designed system, we plan to develop a classification system, as well as a system that is capable of detecting the reception and transmission of a file in real time during network transmission.

Author Contributions: Conceptualization, S.K., S.Y. and D.S. (Dongkyoo Shin); Funding acquisition, D.S. (Dongkyoo Shin); Methodology, S.K., S.Y., H.O., D.S. (Dongil Shin) and D.S. (Dongkyoo Shin); Software, S.K., S.Y. and D.S. (Dongkyoo Shin); Supervision, D.S. (Dongkyoo Shin); Validation, H.O. and D.S. (Dongil Shin); Writing—original draft, S.K. and S.Y.; Writing—review & editing, D.S. (Dongkyoo Shin). All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Defense Acquisition Program Administration, and in part by the Agency for Defense Development under Contract UD190016ED.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Romero, J.A. Sustainable advantages of business value of information technology. In *Encyclopedia of Information Science and Technology*, 4th ed.; IGI Global: Towson, MD, USA, 2018; pp. 923–929.
2. Ha, J.H.; Lee, T.J. Research on text mining based malware analysis technology using string information. *J. Internet Comput. Serv.* **2020**, *21*, 45–55.
3. Westrum, R. Vulnerable technologies: Accident, crime and terrorism. *Interdiscip. Sci. Rev.* **1986**, *11*, 386–391. [[CrossRef](#)]
4. Upchurch, J.; Zhou, X. Malware provenance: Code reuse detection in malicious software at scale. In Proceedings of the 2016 11th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, Puerto Rico, 18–21 October 2016; pp. 101–109.
5. Jeong, K.S.; Bae, S.; Kim, H. Evaluation criteria for suitable authentication method for IoT service provider in industry 4.0 environment. *J. Soc. Korea Ind. Syst. Eng.* **2017**, *40*, 116–122. [[CrossRef](#)]
6. Kang, B.J.; Han, K.S.; Im, E.G. Malicious code trends and detection technologies. *Commun. Korean Inst. Inf. Sci. Eng.* **2012**, *30*, 44–53.
7. Islam, R.; Tian, R.; Batten, L.; Versteeg, S. Classification of malware based on string and function feature selection. In Proceedings of the 2010 Second Cybercrime and Trustworthy Computing Workshop, Ballarat, VIC, Australia, 19–20 July 2010; pp. 9–17.
8. Nataraj, L.; Yegneswaran, V.; Porras, P.; Zhang, J. A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In Proceedings of the AISec, New York, NY, USA, 21 October 2011; pp. 21–30.
9. Chen, T.; Mao, Q.; Lv, M.; Cheng, H.; Li, Y. DroidVecDeep: Android malware detection based on Word2Vec and deep belief network. *KSII Trans. Internet Inf. Syst.* **2019**, *13*, 2180–2197.
10. Naem, H.; Guo, B.; Ullah, F.; Naeem, R.M. A cross-platform malware variant classification based on image representation. *KSII Trans. Internet Inf. Syst.* **2019**, *13*, 3756–3777.
11. Lee, W.; Kim, H. A study on generic unpacking using entropy of opcode address. *J. Digit. Contents Soc.* **2014**, *15*, 373–380. [[CrossRef](#)]
12. Jeong, G.; Choo, E.; Lee, J.; Bat-Erdene, M.; Lee, H. Generic unpacking using entropy analysis. In Proceedings of the 2010 5th International Conference on Malicious and Unwanted Software, Nancy, Lorraine, 19–20 October 2010; pp. 98–105.
13. Woo, C.; Ha, K. A development of malware detection tool based on signature patterns. *J. Korea Soc. Comput. Inf.* **2005**, *10*, 127–135.
14. Shijo, P.V.; Salim, A.J.P.C.S. Intergrated static and dynamic analysis for malware detection. *Procedia Comput. Sci.* **2015**, *46*, 804–811. [[CrossRef](#)]
15. VX Heaven. Vx Heaven Virus Collection 2010-05-18. Available online: <http://vxheaven.org/> (accessed on 9 November 2018).

-
16. Scikit-Learn. Available online: <https://scikit-learn.org/> (accessed on 28 December 2020).
 17. Microsoft Malware Classification Challenge (Big 2015). Available online: <https://www.kaggle.com/c/malware-classification/> (accessed on 9 November 2018).
 18. Tensorflow. Available online: <https://tensorflow.org/> (accessed on 28 December 2020).