MalDC: Malicious Software Detection and Classification using Machine Learning

Jaewoong Moon[1], Subin Kim[1], Park Jangyong[1], Jieun Lee[1], Kyungshin Kim[2], and Jaeseung Song[1*]
[1] Sejong University
209, Neungdong-ro, Gwangjin-gu, Seoul KR
[e-mail: jwmoon10@gmail.com, spdlqj99099@gmail.com, jkff1@naver.com, love9ly@gmail.com, jssong@sejong.ac.kr]
[2] Convergence Technology Collaboration Directorate, Agency for Defense Development, Songpa P.O Box 132, Seoul KR
[e-mail: updatekim@add.re.kr]
[*]Corresponding author: Jaeseung Song

Received March 8, 2022; accepted April 4, 2022; published May 31, 2022

Abstract

Recently, the importance and necessity of artificial intelligence (AI), especially machine learning, has been emphasized. In fact, studies are actively underway to solve complex and challenging problems through the use of AI systems, such as intelligent CCTVs, intelligent AI security systems, and AI surgical robots. Information security that involves analysis and response to security vulnerabilities of software is no exception to this and is recognized as one of the fields wherein significant results are expected when AI is applied. This is because the frequency of malware incidents is gradually increasing, and the available security technologies are limited with regard to the use of software security experts or source code analysis tools. We conducted a study on MalDC, a technique that converts malware into images using machine learning, MalDC showed good performance and was able to analyze and classify different types of malware. MalDC applies a preprocessing step to minimize the noise generated in the image conversion process and employs an image augmentation technique to reinforce the insufficient dataset, thus improving the accuracy of the malware classification. To verify the feasibility of our method, we tested the malware classification technique used by MalDC on a dataset provided by Microsoft and malware data collected by the Korea Internet & Security Agency (KISA). Consequently, an accuracy of 97% was achieved.

Keywords: artificial intelligence, machine learning, malware classification, MalDC

1. Introduction

In recent years, with the rapid dissemination of the Internet and improvement in the performance of computers, different types of software are being used in various areas of human life; however, the side effects of using computers are also increasingly rapidly. For example, in the case of service disruption through distributed denial-of-service (DDoS) attacks and leakage of private information through hacking, users experience various types of damages. Most of such hacking attacks are due to malware attacking vulnerabilities that exist in normal software, and the subsequent damage to users continues to increase every year. According to "The 2014 Internet Intrusion Response Plan" of the Korea Internet Security Center, Korea Internet & Security Agency (KISA), the average daily appearance of malware increased from 1,435 types in 2013 to 8,847 types in 2014, which is an approximate six-fold rise over a year. Furthermore, according to AVTEST, an IT security research organization, the number of malware incidents reported annually has increased by ~12 times in ten years from 99.71 million cases in 2012 to 1.18063 billion cases in 2021 [1, 2].

One of the main reasons for the sharp increase in software-infecting malware is the use of automation tools by hackers for the rapid creation of similar or variant malware. Malware created using automated tools change only the encrypted code part or perform entry point obscuring (EPO) attacks that change the code at the entry point part of the software, thereby changing only the form of software although the type of malicious behavior remains the same. Based on this, they avoid detection by vaccines. The number of similar and variant malware is surging due to the spread of the software-modification-based polymorphic attack method, and techniques to interfere with the analysis of software are also being developed [3].

Machine learning (ML) is a field of artificial intelligence (AI) [4] and refers to a technique that trains the computer using data so that it can create new rules on its own and make decisions without any help from humans. In other words, ML involves the creation of sophisticated algorithms to analyze learned data in order to find and learn specific patterns. ML is used in different areas in real life, such as autonomous vehicle control, various services based on natural language processing, and robotic surgery. In particular, ML has shown excellent results when solving certain problems by learning images. For example, ML can be used to detect dangerous objects or terrorist threats in advance through the analysis of images captured on a closed-circuit television (CCTV) and to restore the pixels of old photographs or the colors of black and white photographs [5, 6].

Attempts to use ML in the field of information security have also been made in recent years. With the advancement of various information and communication technologies, hacking techniques are becoming increasingly sophisticated and complex, causing more damage than before. In previous cyberattacks, intrusion detection and attack analysis could be performed by just using the pattern matching method based on the analysis of conventional attack patterns [7]. Currently, however, the simple pattern matching method alone can no longer be used to detect advanced cyberattacks because of the use of various smart devices and software. Furthermore, cases of cyberattacks using AI have been recently identified, and the defense strategy using some white hat experts is no longer sufficient to block advanced intelligent cyberattacks. Therefore, new technology should be developed by applying AI to the field of information security, such as network intrusion detection, malware analysis, and vulnerability analysis, to respond to intelligent cyberattacks, which are difficult to detect by experts or conventional hacking defense tools.

In the case of cyberattacks using malware, many recent attacks have used concealed malware to collect and leak important data by bypassing security systems to prevent users from being aware of it. Malware poses a major threat to the security of computer systems. As malware becomes increasingly complex and large, it has become increasingly difficult to deal with malware [1]. Because of the rapid spread of malware, the number of signature patterns found by analyzing the patterns of new malware released every year is rapidly increasing. The frequency and processing capability of malware incidents have exceeded human limits. Previously, static analysis and dynamic analysis were used to identify malware hidden inside normal software. However, because conventional analysis methods execute code for analysis, they have a limitation in that the analysis is not effective in terms of the time efficiency or due to code obfuscation. Recently, studies on the development of relevant technology, through the application of AI to the field of malware detection, are being actively conducted [8].

In this study, we propose a malicious software detection and classification (MalDC) framework to classify malware with the same or similar attack patterns. To this end, we constructed a learning dataset by representing the malware in the form of images and trained the ML to use image analysis techniques. Note that our method showed good performance [9]. The developed ML algorithms and learning dataset can be used to discover newly appearing malware easily and quickly to prevent or minimize the damage caused by hacking. For the successful performance of ML, high-quality learning data are required. Therefore, in this study, we used the Kaggle data provided by Microsoft [10] and malware data collected by KISA [11] as the learning data and succeeded in effectively classifying the malware. Furthermore, to obtain the learning dataset sufficiently and remove noise, training was performed using the image dataset and the fine-tuning method of a convolutional neural network (CNN)-based inception model was used [12, 13]. Further, the algorithm's accuracy was improved by sufficiently increasing the image dataset for learning using a data augmentation method. Based on these techniques and procedures, a malware classification success rate of 97% was finally achieved.

The research contributions of this study are as follows:

- A system architecture was designed and developed to classify malware by training the ML model with malware converted into images.
- Techniques were developed to convert malware into images and increase the amount of training data.
- MS Kaggle and the malware dataset provided by KISA were used to develop the ML model and validate the effectiveness of the malware classification technique.

The remainder of this paper is organized as follows. Section II reviews related work and explains the concept and technology of CNN models—a typical ML method—as background knowledge to improve the understanding of the proposed technique. Section III introduces the proposed ML-based malware classification framework. Then, Section IV focusses on the technique of converting malware into images and the method of expanding the learning dataset. Section V provides the analysis and results for the malware classification experiments using the MS Kaggle and KISA malware datasets. Finally, Section VI presents the conclusions and discusses the future research direction.

2. Background and Related Work

This section introduces the basic ML and image classification techniques required to understand the proposed ML-based malware classification method and describes existing studies related to malware classification.

2.1 ML

ML refers to the computing capability that facilitates learning without explicit programming. It is mainly used for classification and prediction, and the training of the ML model is performed using training data. Furthermore, it requires an exact target variable, i.e., explicit reaching point, from a certain domain (predicted dependent variable). Therefore, results vary depending on the relationship between the type and form of input data and the target variable [14].

ML refers to the task of finding the optimal parameter values to most accurately predict the solution for a particular problem. In general, therefore, ML aims to minimize errors for new samples that are not in the training dataset after going through three stages of training, validation, and testing. The following shows brief definitions of the three stages of ML applied in this paper.

- **Stage 1 (Training)**: It is the stage where an optimal state is reached while improving the performance and is called learning or training. This stage is highly affected by the performance of the computer system used.
- **Stage 2 (Validation)**: It is the stage where classification using the CNN is performed after the training is complete. Individual files can be used, or a bundle of multiple files can be used.
- **Stage 3 (Test)**: The classification accuracy at the file level in the validation stage and the statistics obtained from various processes are summarized. It is a process of predicting the target value for a "new" sample that is not in the training dataset. In this stage, data that have new samples are called the test dataset, and the properties that have high performance for the test dataset exhibit a generalization ability.

2.2 Convolutional Neural Network (CNN) Model

CNNs are mainly used in the fields of image recognition, image processing, and computer vision. A CNN is a neural network model that performs an additional image preprocessing task referred to as convolution [14, 15]. While conventional image learning methods provide raw pixel values of images to learn each feature, a CNN receives an original image as-is and learns the features of that image while maintaining its spatial/local information. CNN models have evolved continuously, and several different types of CNN architectures have been developed. A CNN focuses on and utilizes parts of the image rather than the entire image as well as the relationship between the pixels that constitute the image. It has been used in several studies owing to its desirable performance in image pattern analysis. ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a challenge competition that provides a representative large-scale classification dataset and provide brief descriptions of the four different CNN architectures that have shown excellent performance in the literature.

- AlexNet: This architecture was proposed in 2012 and has shown an improved recognition accuracy compared to all existing ML and computer vision approaches. It is a breakthrough method in the fields of ML and computer vision for visual recognition and classification tasks and has sparked a surge of interest in deep learning [16].
- VGGNET: This method was introduced in 2014 and demonstrated that the depth of the network is an important factor affecting the recognition accuracy of a CNN. The VGG architecture has a deeper structure than the AlexNet and uses the ReLU activation function. It also uses a single maximum pooling layer and several fully connected ReLU activation functions. The model's final layer is a SoftMax layer, which performs the classification.

Essentially, after stacking convolutional layers of 3×3 in overlaps, the ReLU functions are used.

- ResNet: This architecture was introduced in 2015 and consists of deeper layers that the ones mentioned above. Variants of the ResNet model have been developed with different depths of layers, such as 24, 50, 101, 152, and 1,202 layers. Among them, the most popular, ResNet-50, consists of 49 convolutional layers and one fully-connected layer. To solve the problem of learning becoming more complex as the depth of the network increases, a concept called skip connection was proposed in the ResNet, which caused a reduction in the complexity [17].
- Inception: Inception is an architecture that has evolved several times from V1 called GoogLeNet in 2014 to V4 developed in 2015. As the learning architecture commonly has a greater width and depth, the performance of this model is the most desirable of the ones mentioned above. However, during training performed, problems such as overfitting occur, and it can be said that a greater width and depth are not always beneficial for an improved performance. Inception successfully implemented an architecture that could lead to maximum performance without compromising performance [3].

2.3 Image Classification Algorithm

Classifying objects in the form of images into available categories is one of the common functions of computer vision. Image classification can be performed using Histogram of Oriented Gradients (HOG) or Scale-Invariant Feature Transform (SIFT), and if there are defined categories, the classifier is used to classify the input image as one of the appropriate categories [19]. In recent years, feature extraction and classification have been performed by a single deep learning classifier model, which showed better accuracy than conventional image classification methods. In particular, ML is used in various areas related to image classification. Esteban et al. proposed AmoebaNet-A, an image classification CNN that, for the first time, surpassed the designs proposed by humans. (Later on, improved versions, AmoebaNet-B, C, and D were additionally proposed.) By introducing an age property to favor the younger genotypes, they modified the evolutionary algorithms applied in tournament selection. AmoebaNet-A discovered parameters and network architectures using more complex architecture search methods. Furthermore, it had comparable accuracy to the latest ImageNet models while maintaining similar parameter sizes; furthermore, in an AmoebaNet architecture of a larger size, it performed with a top-5 error rate of 3.4% [20, 21].

Kaiming et al. developed a network called ResNet that achieved a top-5 error rate of 3.57% in ImageNet tasks with an ensemble structure by stacking 152 layers—more than eight times deeper than the existing VGG16 network. The authors proposed a concept called skip connection and a network of deep structure. AlexNet, VGGNet, and GoogLeNet transformed feature maps through convolutional layers, but ResNet is trained with convolution operations using the values added to the inputs. ResNet with an architecture of up to 152 layers was proposed, and later, a deep architecture with 1,002 layers was announced [22]. ResNet's CNN has been used variously as a basic CNN in the object detection field and showed superior performance in faster-RCNN-based detectors [22].

In addition, Howard et al. developed a MobileNet method based on a streamlined architecture using convolution layers divided in the depth direction. MobileNet was different from common convolution methods in that convolution was applied to each channel, after which a 1 × 1 convolution was applied for operations between the channels. In other words, this can be viewed as a convolution for each channel + a factorized convolution between 1 × 1 channels, whereas it is a common operation to apply the convolution to the whole. This method was proven to be favorable in terms of parameters and speed compared to conventional methods [23]. Furthermore, Huang et al. developed DenseNet, a ResNet-based variant. ResNet applies the skip connection to the next layer only, but DenseNet has a structure that applies the skip connection to all layers. DenseNet has the following advantages: it (1) alleviates the gradient vanishing problem, (2) strengthens feature propagation, (3) encourages feature reuse, and (4) reduces the number of parameters [24].

2.4 Research on Application of Malware-related AI

Because the effectiveness of manual inspection is poor compared to the high spread rate of malware, studies on AI application to malware analysis are actively underway[25, 26, 27, 28, 29]. A study was conducted at the Berlin Institute of Technology to apply ML to malware's behavior-based analysis [5], and research was conducted at the Swiss German University to detect malware based on ML technology by constructing a sandbox and automatically analyzing the malware's behaviors and generating reports. As such, many organizations have been conducting research to apply AI technology to malware [7, 22, 30].

In particular, many studies have recently used AI and ML to detect malware of EXE and PE types [31]. Nvincea Labs in the U.S. conducted research on malware detection methods using deep neural networks (DNNs). After extracting features, such as string 2D histogram and portable executable (PE) import tables and PE metadata, from malicious/normal programs, they applied a method of learning them using DNNs. Furthermore, the DNN model used in the study consisted of one input/output layer and two or more hidden layers that had 1,024 nodes and showed that malware could be classified through one output produced through the model [20]. This research was conducted at CQVista on a non-signature-based malware detection method that facilitates real-time detection while providing high detection rates by extracting raw features of PE files and applying a decision tree algorithm to the results of calculating the information gain (IG) of the extracted features [21]. Furthermore, research was conducted at Maryland University on a method of detecting malware by training a deep learning model with the byte information of files to detect malware of Windows executable files (.exe). For learning byte information using deep learning, a gate structure that performs two convolutional operations was applied to the CNN, and a structure for normal/malicious software classification was added to conduct the training and classification [23].

3. MalDC: Intelligent Malware Classification Framework Design

This section describes the system architecture of MalDC, a Malicious Software Detection and Classification framework. In this study, we represented malware binaries as images and treated the malware classification problem as a problem of classifying images. Conventional malware analysis and detection techniques must analyze in detail or execute malware at the code level. In the case of MalDC, however, because malware is converted into images and then classified through ML, there is an advantage in that it facilitates accurate classification without execution and detailed analysis of code.

3.1 MalDC Overview

Fig. 1 shows the architecture of the MalDC system. MalDC is executed using a binary malware dataset as an input. The binary malware dataset consists of a training dataset used to train the model and a testing dataset used to validate the model. The training data are converted into images to train the ML model. Noise is removed from the created malware images, which then moves through the preprocessing process to increase the training effect. The malware images for training are used as inputs for the training of the ML model for malware classification, based on which the initial model is developed. The performance of the developed malware classification model is assessed using the testing malware dataset. If a certain level of performance is not achieved, a process of modifying the model is performed by retraining the model after changing the parameters, and testing is performed again. If a certain level of performance is achieved, a model to be used in MalDC is finally produced. The developed malware classification can be used to classify existing malware or check if the software suspected of being malware is indeed malware. The following section describes the stages performed repeatedly in the proposed system.

- Imaging Malware: MalDC visualizes malware binaries as gray-scale images. Patterns that show edges of the objects existing in an image are required when learning images, and gray-scale images consist of brightness values without color information. Therefore, ML algorithms can check the patterns of the objects included in gray-scale images faster and more efficiently, compared to those in color images. In case of many malware families and variant families, images belonging to the same family look very similar in terms of layouts and textures. Therefore, MalDC uses a classification method that uses standard image features, motivated by these visual similarities. Obviously, this classification method has the advantage that direct analysis of source code or execution of code is not required, unlike static and dynamic analysis.
- Image pre-processing: After converting malware into images, it is necessary to remove noise, label images, and expand the image dataset to achieve the best performance in ML. For example, in a study conducted at the University of California, Santa Barbara, ML's performance was improved by obtaining a large dataset by enlarging or reducing the image size or changing the width of images in the same rectangular shape. MalDC also applies several preprocessing processes, such as noise removal and image resizing [20].
- Image data learning and development of an optimal model: MalDC uses a CNN algorithm specialized in image classification to classify images. During the initial implementation, we developed the neural network system using Google's open-source TensorFlow library. Afterward, the TensorFlow Inception library, which was newly released by Google to increase the level of image classification accuracy of basic TensorFlow, was applied, and a malware image classification accuracy of over 90% was achieved.

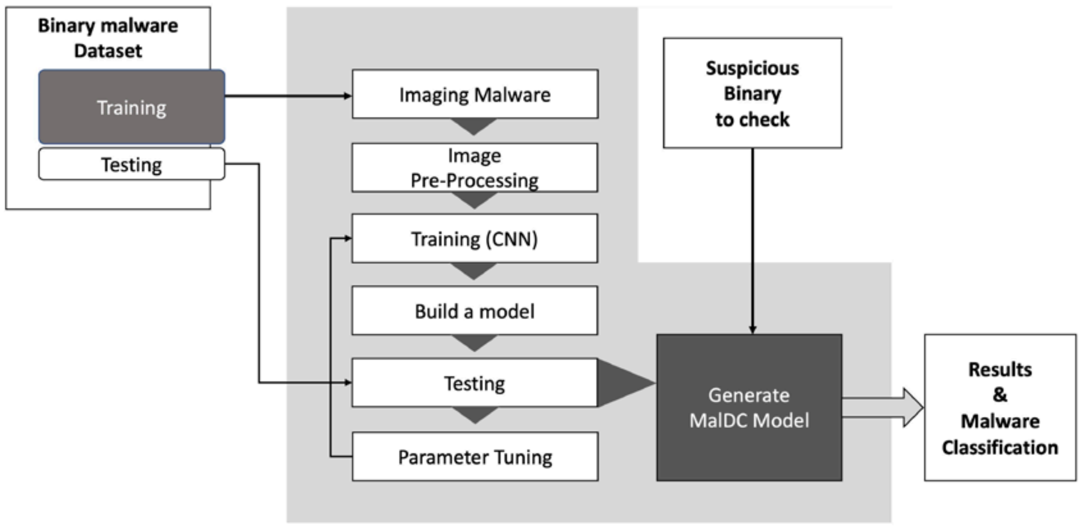


Fig. 1. MalDC framework architecture

3.2 Imaging Malware

Fig. 2 shows two different types of malware visualized as images. In this figure, the images in the first row show three malware instances belonging to the Fakerean family, and the images in the second-row show Dontovo malware-based families [21]. Fakerean is malware that appeared around 2010 and is registered with the Microsoft Malware Protection Center (MMPC). It has the characteristic of disguising itself as an antivirus by forging the name or logo. Dontovo is a kind of Trojan horse, which can install malware or software on computers. Dontovo's core malware code can be ported by hackers to other software. In this case, it is recognized as another software in the external shape, but abnormal activities that actually occur are executed by the malicious code part defined in Dontovo. Because images belonging to the same family have similar visual characteristics, the characteristics of images can be used to classify the family.

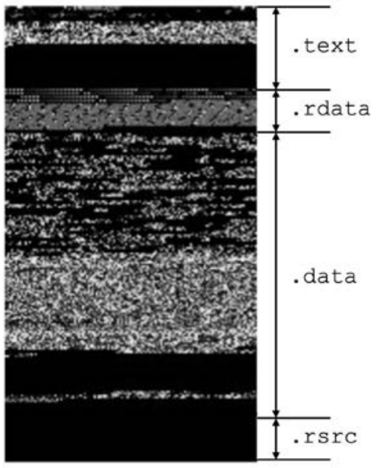


Fig. 2. Various sections that constitute Dontovo A malware

The samples belonging to the same family can be distinguished because they are visually similar. In contrast, images of malware that belong to different families show distinctly different patterns. This is because new malware is created through a method of copying the behavior and features of existing malware and applying them to new software, rather than developing malware from scratch when creating malware. In other words, malware can be classified using computer vision technology that facilitates image-based classification because of the visual similarities of such malware images. In MalDC, malware classification and discrimination are performed based on the image visualization similarities of malware of the same family.

In MalDC, the malware binaries given for the visualization of malware are constructed in a 2D array which is read as vectors of unsigned 8-bit integers. They can be easily visualized in a gray-scale image with a range of [0, 255] (0: black, 255: white). The image's width is fixed, and the height may vary depending on the file size.

Fig. 3 shows a gray-scale image example converted from binaries of downloader Dontovo A, a type of Trojan horse that downloads and executes arbitrary files. In most malware images, several different sections (binary fragments) exist, and each section shows a unique image texture. Dontovo A malware consists of a total of four sections: .text, .rdata, .data, and .rsrc. The .text section has executable code. In **Fig. 3**, the first part of the .text section contains code with a sub-divided texture. The remaining part is filled with zeros, and the end of this part is also filled with zeros. Next, the .data section contains uninitialized code (black patch) and initialized data (detailed texture). The last section is .rsrc, which contains all resources of the module. It can also include an icon that can be used by the application. When malware is represented as an image, various sections of software have different patterns in the image.

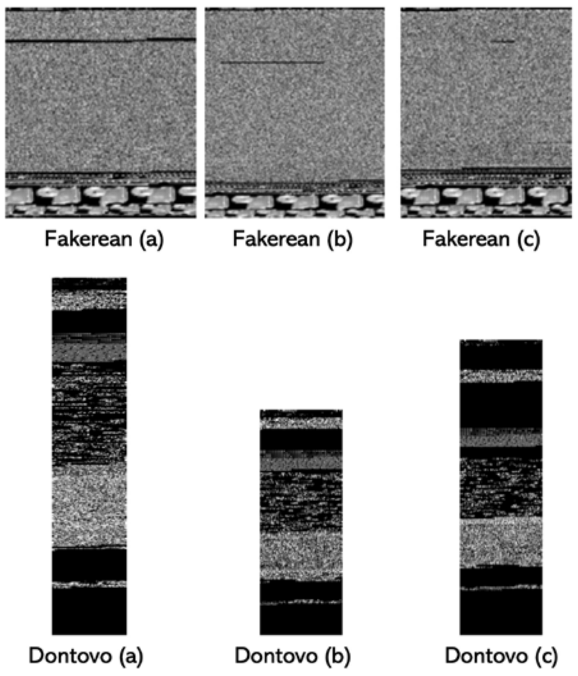


Fig. 3. Classification of malware images by family

3.3 MalDC ML Algorithm and Configuration

To learn and classify images through ML, an appropriate ML algorithm must be selected for the problem to be solved, and various parameters related to the training must be set.
ML Algorithm: We developed a model by selecting Inception, which shows the best performance and is specialized in image classification among ML open-source libraries, to learn and classify malware images using ML in MalDC.

Inception is an ML technique for computer vision developed by Google. GoogLeNet won first with V1 in a global image recognition challenge in 2014, and currently, V4 was released. The field with the highest utilization of Inception technology is the medical image analysis field. Google Health's medical imaging team has revealed that AI developed based on Google's computer vision technology has reached such a level that diabetic retinopathy can be diagnosed using AI. The AI system using Inception diagnosed diabetic retinopathy accurately, recording an F-score of 0.95 (the maximum is 1)—which is slightly higher than 0.91, the average of eight doctors—considering the sensitivity and specificity required for diagnosis. Inception models can be classified into the from-scratch method and fine-tuning method according to the training method [22].

- From-scratch method: As the name suggests, images to be classified are converted into images that can be recognized by ML, and after going through the training process, they are stored in the neural network. This is followed by the classification process, but the accuracy is lower than that of the fine-tuning method, in which the training process is performed once more with the training results.

- Fine-tuning method: The exact meaning of this method can be found in a notation, Fine-Tune a Pre-Trained Model. Weights are adjusted in the existing training models, or a preprocessed dataset is created to make a new training model. It is feasible when classifying desired images, but images cannot be recognized and learned immediately during training. Therefore, they are converted in a way that they can be recognized in ML, and the images and image labels are structured and stored. MalDC uses the fine-tuning method for high accuracy [7].

Parameter settings: In ML, parameters are variables determined by the model, and their values are determined from data. Hyper-parameters refer to values that the user sets by experience when training the model. In MalDC, the following parameters were used.

- Validation: This refers to a proportion of the image learning dataset used for evaluation. The fitness of the ML algorithm is determined in the following manner. Usually, a predetermined percentage of the given labeled image group is used for training, and after using the remaining images for evaluation, the fitness of the algorithm is determined based on the results. For example, if the validation value is set to 100, X – 100 images out of a total of X images are used for training, and the remainder, 100 images are used for evaluation.
- Steps: This is a parameter used in the training stage and provides information that determines the number of steps to go through when training the ML model and the number of images to be used each time when training. This is related to the training time. If this value is large, training takes a long time.
- Batch size: It is a parameter used in both the training and validation stages, and its value determines how many images will be learned or evaluated in each execution step.
- Learning rate: This refers to the size of the search rate used in the training. For example, if the learning rate is too large, overshooting may occur, leading to no learning. In contrast, if the learning rate is too small, the training steps are too small. Thus, even if a very long time is spent on training, learning does not take place, resulting in very poor accuracy. Because there is no recommended value for this, the size of the value was tuned up or down between experiments according to the given input layer's image characteristics, and the accuracy was checked.
- Weight decay: This parameter is used as a supplementary factor for the learning rate. If the learning rate is too small, causing overfitting, one of the solutions may be to make the learning rate smaller, but in some cases, the weight decay value is tuned to suppress overfitting. This method is called regulation. In MalDC, the weight decay between experiments was fixed mostly at 0.00004.

4. MalDC Implementation

This section introduces the development environment settings for the implementation and experiments using MalDC and the core algorithms of major program modules used in MalDC.

4.1 Equations

In MalDC, TensorFlow and Python 3 were used together to execute ML algorithms. **Table 1** shows the development environment required for the construction and operation of MalDC. Furthermore, we developed the modules required for converting malware into images, learning the malware images, and evaluating the trained model for malware classification in MalDC. **Table 2** provides a list of the developed modules and brief descriptions of their functions. All modules were developed using Python. The following section describes the execution logic of each module program.

Table 1. Basic development environment

Windows development environment • CPU: Intel i7-6700HQ @ 2.6GHz, RAM 8GB • Windows 10 home 64bit • GPU: NVIDIA GEFORCE GTX 965M
Linux development environment • CPU: POWER8 processor, 3.32GHz 2-socket 16-core • Memory: 128GB • Ubuntu 16.04.3 ppc64 (Kernel: 4.8.0-36) • GPU: Two units of K80 (a total of four GPUs) • DISK: 1TB x 2

Table 2. Basic development environment

Class	Applications
Imaging Module (MalDC-M1)	Performs the function of preparing a training dataset by downloading and converting malware into TFRecord, the image format for training. Module name: malware_imaging.py
Training Module (MalDC-M2)	Performs the function of developing a model by executing actual ML using the prepared training dataset of malware images. Module name: maldc_training.py
Evaluation Module (MalDC-M3)	The accuracy of the model developed based on the training is measured using the test image dataset. Module name: maldc_eval.py

4.2 MalDC Modules

MalDC-M1: MalDC-M1 is a module that performs imaging of malware binary files in MalDC. It receives a malware dataset as input and converts them into images suitable for ML. Because TFRecord is used as the image format for training in MalDC, MalDC-M1 implements the function of converting malware code into the TFRecord format. Before converting into images of malware, MalDC-M1 first checks for normal operation. Three image sets are used for the initial operation check: cifar10, flowers, and mnist, which are widely used in ML research. Cifar10 is a dataset comprising 60,000 color images, and each image is labeled as one of ten classes (airplane, car, bird, cat, deer, dog, frog, horse, ship, and truck). Flowers is a dataset of flower images classified by five labels. Mnist is a large image database used widely for training and testing in the ML field. It was created by remixing samples of the original dataset of NIST. After checking for normal operation first using regular image sets, conversion into images is performed with the MS Keggle dataset and KISA ransomware dataset.

MalDC-M2: MalDC-M2, a training module, performs image learning using a deep learning model. As mentioned earlier, there are two main methods of training. The first one is the “training a model from scratch” method, which means that it is the bottommost method of image processing. In other words, it is a method of creating a completely new model, not changing a previously developed model. The second one is the “fine-tuning a model from an existing checkpoint” method (simply, the “fine-tuning” method), which creates a model by training an already well-crafted model with additional images. In MalDC, accuracy experiments were conducted for the two methods.

- **From-scratch method**: Training was newly performed with the Inception_V1 model using the converted malware image set. At the beginning, various settings (e.g., model name, learning rate, intermediate storage time, and maximum steps) were configured for model development. Then, after selecting a training dataset, network, and preprocessing task type, if all settings were complete, the model was developed through actual training.
- **Fine-tuning method**: The fine-tuning method is a method of learning new images based on an already created model. When training a “from scratch” model, the Inception_V1 model was used to classify images. In the fine-tuning method, the pre-trained model was downloaded in ImageNet using the same model for comparison, and the model was retrained. Thus, a model that recorded an 89.6% accuracy was achieved, which is ImageNet’s top-5 accuracy. The training process is performed twice in the fine-tuning method. In Process 1, only the last layer is newly trained in the downloaded model, and in Process 2, all layers are trained again in the model that has undergone Training Process 1.

MalDC-M3: This helps select an appropriate ML model for malware image classification by determining the accuracy of the models created in the training module. When a new model is created, it can be used to classify malware. Malware images that were not used in training are provided as a test dataset in the model, and classification is performed. By checking whether the images of malware used in the test were correctly classified, it can verify the accuracy of the predictions (malware classification in this paper) made by the created model. If the accuracy is high in the test results, model development is completed. If the accuracy is not high, the training process is repeated by adjusting the parameters to increase the accuracy.

5. Experiments and Evaluation

This section shows and discusses the experimental results for MalDC. In particular, the adequacy of the method is determined based on accuracy by applying various approaches, such as changing the training method, preprocessing data, and tuning the parameters, and the methods used to improve accuracy are introduced.

5.1 Malware Datasets

The malware datasets applied to the experiments are the Keggle dataset provided by MS, which consists of 10,868 data with nine labels, and a dataset provided by KISA, which consists of 689 data with five labels. Table 3 shows the labels of each dataset and the number of data included for each label. Both the MS and KISA datasets have an imbalance problem in the minimum number of training images and the number of images for each label, which is a hurdle in creating high-accuracy models.

Table 3. Information of MS Keggle and KISA Ransomware

Dataset Type	Label	Quantity	Remark
Microsoft Keggle Dataset	1	1,541	
	2	2,478	
	3	2,942	Label with maximum data
	4	475	

	5	42	Label with minimum data
	6	751	
	7	398	
	8	1,228	
	9	1,013	
	Total	10,868	
KISA Ransomware Dataset	1	293	Label with maximum data
	2	131	
	3	23	Label with minimum data
	4	154	
	5	88	
	Total	689	

MS's dataset consists of 10,868 malware data for training, which is small number for training of ML. Even MNIST, the first practical dataset for ML, consists of more than 20,000 handwriting images. However, more important factors that affect accuracy than the number of data are the dataset's quality and the imbalance problem between labels.

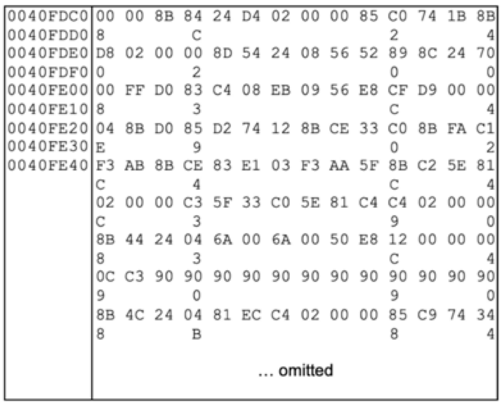
In other words, the quantity of data to be learned, the quality of data, and the balance between the labels in the dataset are the most important factors that determine the accuracy of the image learning/classifier; however, the MS Keggle dataset and KISA's dataset do not satisfy all these three factors. Since the limitation in the amount of data is unavoidable due to characteristics of malware, unlike regular photographs or facial images, we placed the focus of the MalDC experiments on finding the model and method that can obtain as high an accuracy as possible.

In the case of MS Keggle dataset, Label No. 5 consists of a total of 42 files, as shown in **Table 3**. There are 2,942 files in the case of Label No. 3, which has the most files, followed by 2,478 files for Label No. 2. Compared to these, the fact that the label with the minimum data consists of 42 files means that there is a serious imbalance. The imbalance has a very significant effect on the accuracy of image classification. Even if the images of Label No. 5 have accurate signatures, it is not easy to perform accuracte classification with only 42 training data.

5.2 Quality of Datasets

The data in the MS Keggle dataset used in this study are all executable files of the Portable Executable (PE) format that collected malware. PE files are Windows execution data representing bytes in the hexadecimal format, and the data contains many undefined bytes (represented as ??) depending on the file characteristics.

When the content of a normal PE file that does not contain undefined bytes is examined, it is represented as **Fig. 4**. It shows the content of the 0akIgwhWHYm1dzsNqBFx.bytes (Label No. 2) file, one of the malware images. As such, all contents in a normal malware image file are in normal hexadecimal format. In contrast, there are files that consist of undefined code (??), which are impossible to represent as hexadecimals.

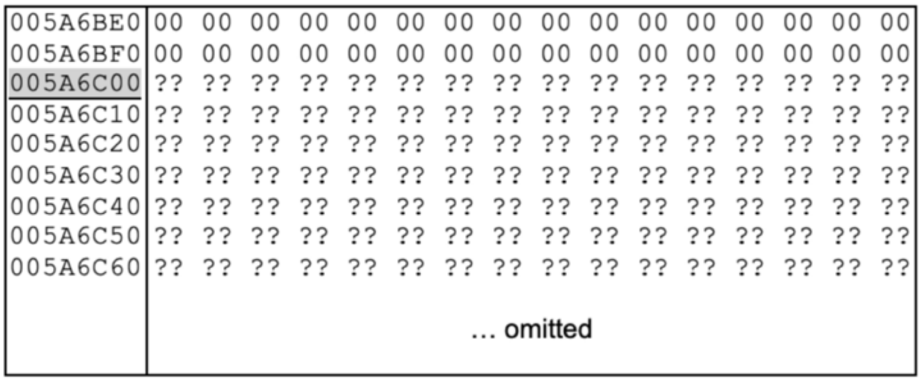


```
0040FDC0 00 00 8B 84 24 D4 02 00 00 85 C0 74 1B 8B
0040FDD0 8        C                    2         4
0040FDE0 D8 02 00 00 8D 54 24 08 56 52 89 8C 24 70
0040FDF0 0        2                    0         0
0040FE00 00 FF D0 83 C4 08 EB 09 56 E8 CF D9 00 00
0040FE10 8        3                    C         4
0040FE20 04 8B D0 85 D2 74 12 8B CE 33 C0 8B FA C1
0040FE30 E        9                    0         2
0040FE40 F3 AB 8B CE 83 E1 03 F3 AA 5F 8B C2 5E 81
         C        4                    C         4
         02 00 00 C3 5F 33 C0 5E 81 C4 C4 02 00 00
         C        3                    9         0
         8B 44 24 04 6A 00 6A 00 50 E8 12 00 00 00
         8        3                    C         4
         0C C3 90 90 90 90 90 90 90 90 90 90 90 90
         9        0                    9         0
         8B 4C 24 04 81 EC C4 02 00 00 85 C9 74 34
         8        B                    8         4

                   ... omitted
```

Fig. 4. Byte structure of normal malware

Fig. 5 shows the content corresponding to the lower part of the image file of the above malware. The far left side of the figure shows the memory dump address part, and to the right is a list of 16-byte virus codes. **Fig. 5** shows that the contents of all byte codes from the address 005A6C00 to the bottom consist of ??. As such, the question mark (??) part is displayed in the hexadecimal window when represented as byte code because the virtual address range of values corresponding to a certain range cannot be known. In the file structure, it is commonly referred to as the BSS section, and it is used to reflect the static storage requirement of the program, although the space is not usually secured in the file. Here, the BSS section is a place created to place the static variables or a part of the program that the compiler did not initialize. When we checked whether the BSS section represented as ?? was contained in 10,868 files of the MS Keggle dataset, we found that BSS sections of over 90% were contained.



```
005A6BE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005A6BF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005A6C00 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
005A6C10 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
005A6C20 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
005A6C30 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
005A6C40 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
005A6C50 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
005A6C60 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??

                    ... omitted
```

Fig. 5. Byte structure of abnormal malware with the BSS section

Table 4 shows the result of investigating whether ?? bytes are contained in 10,868 malware files. **Table 4** is a statistics table showing how many malware files contain ?? bytes at the label level from Label No. 1 to 9. For example, a total of 398 files exist in the case of malware for Label No. 7, and 323 files contain over 90% of ??. If over 90% is ?? bytes, it is difficult to expect high accuracy for this file, irrespective of which image learning and classification system is used. Therefore, to obtain more accurate classification results in MalDC, we improved the accuracy by tuning various libraries, methods, models, or various parameters. For the images for quality improvement method, the next section describes the experimental method in more detail.

Table 4. Inclusion of the BSS section

Label	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Average(%)	7	2	63	41	69	9	83	23	2
0%	1213	2384	5	23	1	629	38	836	948
10%	270	64	0	28	2	27	0	24	38
20%	21	3	0	45	2	15	13	16	13
30%	10	3	0	69	0	32	18	15	0
40%	9	3	317	130	1	10	5	25	1
50%	1	0	0	177	6	13	1	70	6
60%	1	16	2437	3	5	19	0	90	3
70%	6	2	183	0	7	3	0	134	2
80%	0	1	0	0	14	2	0	18	2
90%	2	2	0	0	4	1	323	0	0
100%	8	0	0	0	0	0	0	0	0
Sum	1541	2478	2942	475	42	751	398	1228	1013

5.3 Training Comparison Between Fine-tuning and From-scratch methods

In the preceding section, we explained the difference between the fine-tuning method and the from-scratch method before experimenting with the fine-tuning method. The from-scratch method is a training method of basic CNN, which learns images and evaluates new images based on the results, whereas the fine-tuning method has emerged recently to increase the accuracy of image learning using such a neural network.

The fine-tuning method refers to a method for learning the content learned previously again by leaving a checkpoint while leaving the learning results with the neural network. In general, the accuracy of the fine-tuning method is high; however, in some cases, the from-scratch method has a higher accuracy depending on the image's shape or complexity. In this study, therefore, we tested both methods and measured the accuracy. This can be viewed as an experiment for determining which method—the fine-tuning or from-scratch method—is more suitable for malware image classification.

Table 5 shows various experimental results related to this. In the first experiment, the same values were given for all parameters, except for the method, and the from-scratch and fine-tuning methods were applied. In this case, the fine-tuning method showed 52% accuracy, which was slightly higher than that if the from-scratch method of 50%.

Table 5. Comparative analysis of the fine-tuning and from scratch methods

	Experiment 1 (Same Parameters)		**Experiment 2 (Changing Weight Decay)**		**Experiment 3 (Changing Batch Size)**		**Experiment 4 (Changing Steps)**		
Category	Case 1	Case 2	Case 1	Case 2	Case 1	Case 2	Case 1	Case 2	Case 3
Method	Fine tune	Scratch	Fine tune	Scratch	Scratch	Scratch	Scratch	Scratch	Scratch
Validation	800/500	800	1000/500	1000	500	500	1500	1500	1500
Steps	1000	1000	1000	1000	1500	1500	1500	1500	2600
Batch size	32	32	16	16	16	32	16	16	32

Learning rate	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Weight decay	0.00 004	0.00 004	0.0000 4	0.0000 1	0.00 004	0.00 004	0.00 004	0.00 004	0.00 004
Result	**52%**	50%	**56%**	54%	56%	**57%**	**66%**	54%	57%

In the second experiment, images were learned and evaluated by changing only the weight decay (fine-tuning method: 0.00004 / from-scratch method: 0.00001), one of the important parameters in ML, under the same conditions as those used in the first experiment. In the accuracy results of the second experiment, both methods showed an improvement in accuracy compared to the results of the previous experiment. In contrast, the difference in the environmental variable did not result in a big difference in accuracy between the two methods. Like the first experiment, the fine-tuning method showed a accuracy of 56%, which was ~2% higher than 54% of the from-scratch method.

The third experiment was conducted to examine the change in accuracy by changing the parameters in the same method. That is, in the experiment of other parameters for the from-scratch method, the accuracy was examined according to batch size. In Cases 1 and 2, the batch size was set to 16 and 32, respectively, and the accuracy was found to be slightly higher when the batch size was larger.

In the fourth experiment, too, the change in accuracy was measured while tuning the Steps parameter in the same from-scratch method. The accuracy was measured by changing the Steps to 1,000, 1,500, and 2,600, while all other parameters were fixed. As the value of Steps increased, accuracy decreased, and an accuracy of 66% was recorded when the value of Steps was 1,000, showing the highest value among experiments 1 through 4.

In the next section, we detail experiments conducted to measure the accuracy according to the image shape.

5.4 Experiments Based on Image Shape

The experiments in the preceding section evaluated the accuracy based on the method of learning images and tuning of various parameters. The from-scratch method had the highest accuracy of 66% when the value of Steps was set to 1,000. However, since it is impossible to classify malware adequately with a model with an accuracy of 60%, we examined other methods that can increase accuracy. First, we experimented with the shape of imaging malware. In other words, we compared a method of processing with the square shape regardless of the image size and a method of processing with a rectangular shape of a certain size (e.g., width is set to 1,024 pixels).

Table 6 shows the results of comparative experimentation with the accuracy after converting the image's shape into a square and a rectangle in the process of converting malware binaries. The model that was evaluated after learning the images of malware converted into squares showed very little difference from the model evaluated after creating rectangular images of malware: both showed an accuracy of ~65 to 66%. Based on these results, we determined that the method used to create the first image with malware has no significant impact on the accuracy of classification. Therefore, all experiments, thereafter, were conducted by converting malware into square shapes in the fine-tuning method.

Table 6. Accuracy comparison between square and rectangle malware images

	Case 1	**Case 2**	**Note**
Type	Square	Rectangle	Rectangle
Methode	Scratch	Scratch	Scratch
Validation	1500	1500	1500
Steps	1000	1000	1000
Batch size	16	16	16
Learning rate	0.01	0.01	0.01
Weight decay	0.00004	0.00004	0.00004
Result	66%	65%	66%

5.5 Accuracy Improvement of the Model

Although we decided to convert malware into square images in the future, because the accuracy shown in the experiments conducted so far remained in the 60% range, a method that could improve the classification accuracy of the model was required. Because the BSS section, i.e., the bytes represented as ??, in malware accounts for a significant portion, it is not easy to achieve an accuracy of 90% or more.

In the case of BSS sections, the problem can be solved to some extent by replacing the ?? bytes with a specific color. The following explains each method.

- Conversion into black: The black method inserts 0x00 in place of the ?? bytes. If this method is applied, files with many ?? are mostly displayed as black. As shown below, files in which almost all bytes are displayed as black correspond to a case in which most of the existing malware consists of ?? bytes. However, if the proportion of the section containing ?? is small, it can be used as a signature.
- Conversion into white: The white method inserts 0xFF in place of?? bytes. This method has a relatively more efficient characteristic than the black method because black (0x00) is included more than white in the static analysis phase of malware.
- Conversion into specific bits: The specific bit conversion method inserts a certain bit pattern for each label instead of replacing the ?? byte parts with black or white. For example, 0x10 is inserted in the ?? parts of the malware having Label No. 1, and 0x20 is inserted in malware of Label No. 2. Usually, patterns such as lines and planes are considered more important than the color of the corresponding parts during the processes of learning and identifying images. Therefore, the bit conversion method is likely to contribute to improving the accuracy of image classification.

However, it is difficult to expect high accuracy when the quantity of data in the training dataset is small. To improve the image learning accuracy with a dataset having a small number of data, the dataset was quantitatively increased by applying a data augmentation technique. The data augmentation technique creates many converted images with the same label by applying various methods of image conversion algorithms to the original labeled images. For example, an image labeled as an apple is rotated to the right by 90° to create a total of four images. In this study, we applied the following four data augmentation techniques: crop, rotate, invert, and scale (CRIS) data augmentation techniques.

- Crop: A part of an image is cropped. It is important to crop some pixels on the side so that the core part of the image is not lost.
- Rotate: The image is rotated by a certain angle. In this study, it was rotated by 90° at a time to prevent the loss of the core part.

- Invert: The image is inverted; this process is also called flip. By flipping left to right or top to bottom, data can be obtained without loss of pixels.
- Scale: It is a method of enlarging or reducing the image size. Similar to crop, it is important to enlarge or reduce in a way that the core part is not lost.

To supplement the part where accuracy does not improve due to the existence of BSS sections, the dataset collected by KISA was used for CRIS data augmentation. As described in Section 5.1, the KISA dataset did not have any ?? bytes of the BSS section which was present in the MS Keggle dataset. This provides good conditions for evaluating accuracy improvements achieved through data augmentation or parameters without the influence of the BSS section. However, because the KISA dataset also has a small number of files for training—less than 700—and a severe imbalance between labels, it is not easy to achieve high accuracy with conventional ML methods.

In this study, we used CRIS data augmentation to expand the dataset from a total of 689 images to 7,000 images. To examine the relationship between the CRIS method and the insufficient dataset and find model training methods and strategies that can increase accuracy, we conducted experiments using various methods. **Table 7** shows the results of several experiments conducted using the KISA ransomware dataset.

Table 7. Experimental results for applying CRIS data augmentation method

	Experiment 1	**Experiment 2**	**Experiment 3**		
Cases	Case 1	Case 1	Case 1	Case 2	Case 3
Method	Fine tune	Fine tune	Fine tune	Fine tune	Fine tune
Validation	70	700	700	700	700
Steps	1000 / 500	1000 / 500	1000 / 500	5000 / 3000	8000 / 4000
Batch size	64	64	64	64	64
Learning rate	0.01 / 0.001	0.01 / 0.001	0.01 / 0.001	0.01 / 0.001	0.01 / 0.001
Weight decay	0.00004	0.00004	0.00004	0.00004	0.00004
Model	InceptionV1	InceptionV1	InceptionV1	InceptionV1	InceptionV1
Result	**50%**	82%	82%	82%	82%

In the first experiment, 689 malware files in the KISA dataset were not converted at all and were applied to the TensorFlow library for learning images. The values of the parameters applied in the experiment for training were as follows: batch size: 64; Steps: 1,000/500; and learning rate: 0.01. When 70 images in the test dataset were classified, an accuracy of ~50% was obtained, as expected. This was not very different from the accuracy obtained by the MS Keggle dataset that contained BSS sections.

In the second experiment, the number of images for ML was increased to check if the accuracy had improved. Here, the CRIS data augmentation technique was applied to increase the insufficient number of images. By applying the CRIS technique, the original images were expanded to ~7,000 images with the same patterns. In the dataset expanded to a total of 7,000 images, 10%—700 images—were allocated for validation. The parameters applied in the preceding experiment (64 for the batch size, 1,000/500 for Steps, and 0.01 for the learning rate) were used for training, and the validation results showed that the image classification accuracy increased to 82%.

The third experiment was conducted for three cases of changing parameters for the dataset expanded to 7,000 images using the CRIS technique. Under the same conditions as the second experiment, the experiment was conducted and accuracy was examined while the learning rate was changed to 1,000, 5,000, and 8,000, and to 500, 3,000, and 4,000 in the secondary training. Because imbalanced labels exist in the KISA dataset, we could not achieve an accuracy higher than 82%, which was achieved by increasing the number of data through the CRIS technique. In other words, we drew a conclusion that a solution is required for imbalanced labels.

In the early stage of developing the malware classification and inspection system using ML, such an imbalance problem may occur due to insufficient training data. However, this problem will be solved naturally if many malware data are collected and are used for the system to learn. To examine the accuracy improvement hindrance caused by the imbalance in the labeled dataset, we conducted experiments by additionally excluding the imbalanced labeled data in the dataset.

Previously, **Table 3** shows that the data of Label No. 3 and No. 5 are relatively smaller than those of other labels in the KISA dataset. In other words, the imbalance is severe in the case of Label Numbers. 3 and 5. In the experiment, therefore, we examined the changes in accuracy by performing training and validation after removing the imbalanced labeled data from the dataset.

Table 8 shows the results of experiments conducted by removing the imbalanced data from the KISA dataset. In the first experiment, the data of imbalanced Label No. 3 were removed. That is, this experiment excluded Label No. 3, which has the most severely imbalanced files. In the KISA dataset comprising 689 files, Label No. 3 consists of 23 files. The experiment was conducted by applying the CRIS data augmentation technique to the dataset of 666 files, excluding the malware files corresponding to this label, and as a result, an accuracy of 84% was obtained.

Table 8. Experimental results based on removal of imbalanced labels and application of dynamic cropping

	Experiment 1 (Only CRIS)	**Experiment 2 (removal label 3)**	**Experiment 3 (removal label 3 & 5)**	**Experiment 3 (Dynamic Cropping)**
Method	Fine tune	Fine tune	Fine tune	Fine tune
Validation	600	600	600	600
Steps	1000 / 500	1000 / 500	1000 / 500	1000 / 500
Batch size	64	64	64	64
Learning rate	0.01 / 0.001	0.01 / 0.001	0.01 / 0.001	0.01 / 0.001
Weight decay	0.00004	0.00004	0.00004	0.00004
Model	InceptionV1	InceptionV1	InceptionV1	InceptionV1
Result	82%	84%	87%	97%

Next, the second experiment was conducted by removing imbalanced Label Nos. 3 and 5 at the same time. The dataset of Label No. 5 causes the second-greatest imbalance after the dataset of Label 3. Label No. 5 consists of a total of 86 malware files, accounting for 12% of the total malware files. When Label Nos. 3 and 5 are removed at the same time, a total of 109 malware files are excluded from training. Therefore, the model was trained using 580 malware files consisting of three labels. As a result, the accuracy improved to 87%. This confirms once again that the existence of imbalanced labels greatly reduces the accuracy of the model development through ML.

After recording an accuracy of up to 87% in the above experiment, we added one more experiment shown in **Table 6**. That is, we added an additional image dataset by slightly changing the method used in the CRIS data augmentation. In the conventional CRIS method, a fixed starting point existed when images were processed. However, the method applied additionally in the experiment was a dynamic cropping method, which changes the reference point of the image conversion. In other words, when cropping was performed before, a certain size was cropped at the starting point of the image, but the dynamic cropping method applied in the experiment changes the cropping position to an intermediate arbitrary point. Based on this, 580 malware images obtained after removing Label Nos. 3 and 5 were expanded to ~6,000 images. The accuracy confirmed through the experiment was 97%, the highest accuracy measured in this study. When the CRIS method was applied to malware images, as a result of applying various changes to the cropping method, a very high accuracy was recorded with an extremely small number of malware images for training—580 original files.

6. Conclusion

ML has been in the spotlight as the most feasible solution for defense against malware attacks, which are continuously increasing and becoming more complex. Research is being actively conducted in South Korea and other countries to identify malware using AI and ML. To develop a system capable of classifying malware using ML and even discover new malware, we developed the MalDC system that creates high-quality malware image datasets for training by converting malware into images and then preprocessing the obtained image dataset. MalDC provides a strategy for classifying malware with high accuracy through experiments conducted using the MS Kaggle dataset and the KISA ransomware dataset, thereby verifying the feasibility of the proposed system.

Research on malware analysis using ML requires the consideration of two main factors. First, the method of converting malware into images should be developed. Through experimental results, we found that it is important to determine the width value such that the unique signatures of malware can be visualized when converting malware into images for MalDC. If the most appropriate images can be created by flexibly changing the width, the accuracy of the malware classification using ML will increase. This is because the malicious code exists in a certain part of the malware because a malware is designed to alter regular software and perform malicious activities. Therefore, if information regarding the part where the malicious code is located can be found, a higher accuracy can be achieved. Furthermore, we found that the accurate selection of the locations of four sections constituting a malicious PE file and the method of processing the BSS section are factors that affect the accuracy.

Second, accuracy can be increased by improving the quantity and quality of the malware image dataset. This is generally true for all applications using ML. Because a sufficient amount of malware could not be used to train the ML model in MalDC, we conducted various experiments to solve the insufficient quantity and quality problem of training data. According to our experimental results, the use of the fine-tuning method for learning of malware images led to a relatively high accuracy. The CRIS data augmentation method was used to expand the size of the malware image dataset and secure quality, and a high-quality dataset of 689 malware files without BSS sections provided by KISA was used by removing imbalanced labeled data. As a result, an accuracy of 97% was achieved.

MalDC showed that, unlike conventional static analysis and dynamic analysis, ML can be used to divide the malware into groups based on the same malicious behaviors and ultimately discover new malware. When a malware classification model is trained using ML, the accuracy may vary significantly depending on the data preprocessing method used. In the future, we plan to investigate methods to perform high-quality data augmentation by preprocessing malware images in a more sophisticated manner.

Acknowledgement

This work was supported by Agency for Defense Development (ADD) and Defense Acquisition Program Administration (DAPA). [No. UD210029TD]

References

[1] AV-TEST. Malware, 2021. Article(CrossRef Link)

[2] Z. K. Zhang, M. C. Y. Cho, C. W. Wang, C. W. Hsu, C. K. Chen, S. Shieh, "IoT Security: Ongoing Challenges and Research Opportunities," in *Proc. of 2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, pp. 230–234, 2014. Article (CrossRef Link)

[3] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, "The evolution of android malware and android analysis techniques," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, pp.1-41, Dec. 2017. Article (CrossRef Link)

[4] M. Stephen, "Alan Turing and the development of Artificial Intelligence," *AI communications*, vol. 27, no. 1, pp. 3-10, 2014. Article (CrossRef Link)

[5] H. J. Matthew, A. M. Swiergosz, H. S. Haeberle, J. M. Karnuta, J. L. Schaffer, V. E. Krebs, A. I. Spitzer, and P. N. Ramkumar, "Machine learning and artificial intelligence: definitions, applications, and future directions," *Current reviews in musculoskeletal medicine*, vol. 13, no. 1, pp. 69-76, Feb. 2020. Article (CrossRef Link)

[6] M. I. Jordan, and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255-260, 2015. Article (CrossRef Link)

[7] J. S. Luo, and D. C. T. Lo, "Binary malware image classification using machine learning with local binary pattern," in *Proc. of 2017 IEEE International Conference on Big Data (Big Data)*, IEEE, pp. 4664-4667, 2017. Article (CrossRef Link)

[8] C. F. Tsai, Y. F. Hsu, C. Y. Lin, and W. Y. Lin, "Intrusion detection by machine learning: A review," *expert systems with applications*, vol. 36, no. 10, pp. 11994-12000, Dec. 2009. Article (CrossRef Link)

[9] D. Bruijne, M. Marleen, "Machine learning approaches in medical image analysis: From detection to diagnosis," *Medical image analysis*, vol. 33, pp. 94-97, 2016. Article (CrossRef Link)

[10] R. Royi, M. Radu, C. Feuerstein, Y. T. Elad, and M. Ahmadi, "Microsoft malware classification challenge," 2018. Article (CrossRef Link)

[11] Korea Internet Security Agency, "CISC2017 data challence Malwares 2017," Jan. 2022. Article (CrossRef Link)

[12] X. Jin, J. Chi, S. Peng, Y. Tian, C. Ye, and X. Li, "Deep image aesthetics classification using inception modules and fine-tuning connected layer," in *Proc. of 2016 8th international conference on wireless communications & signal processing (WCSP)*, IEEE, pp. 1-6, 2016. Article (CrossRef Link)

[13] T. Nima, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, and J. Liang, "Convolutional neural networks for medical image analysis: Full training or fine tuning?," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp.1299-1312, May. 2016. Article (CrossRef Link)

[14] O. Simeone, "A very brief introduction to machine learning with applications to communication systems," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 4, pp. 648-664, 2018. Article (CrossRef Link)

[15] D. Simeone, and P. M. Ameer, "Brain tumor classification using deep CNN features via transfer learning," *Computers in biology and medicine*, vol. 111, pp. 103345, Jun. 2019. Article (CrossRef Link)

[16] L. Yann, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436-444, May. 2015. Article (CrossRef Link)

[17] S. Endang, R. Sustika, R. S. Yuwana, A. Subekti, and H. F. Pardede, "Deep structured convolutional neural network for tomato diseases detection," in *Proc. of 2018 international conference on advanced computer science and information systems (ICACSIS)*, IEEE, pp. 385-390, 2018. Article (CrossRef Link)
[18] M. Agnieszka, and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," in *Proc. of 2018 international interdisciplinary PhD workshop (IIPhDW)*, IEEE, pp. 117-122, 2018. Article (CrossRef Link)
[19] H. Mahbub, J. J. Bird, and D. R. Faria, "A study on cnn transfer learning for image classification," in *Proc. of UK Workshop on computational Intelligence*, Springer, Cham, pp. 191-202, 2018. Article (CrossRef Link)
[20] R. Waseem, and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, vol. 29, no. 9, pp. 2352-2449, 2017. Article (CrossRef Link)
[21] R. Esteban, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. of the aaai conference on artificial intelligence*, vol. 33, no. 01, pp. 4780-4789, Feb. 2019. Article (CrossRef Link)
[22] H. Kaiming, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. of European conference on computer vision*, Springer, Cham, pp. 630-645, Oct. 2016. Article (CrossRef Link)
[23] H. Andrew, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," Apl. 2017. Article (CrossRef Link)
[24] H. Gao, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. of the IEEE conference on computer vision and pattern recognition*, pp. 4700-4708, 2017. Article (CrossRef Link)
[25] N. F. Amalina, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," *Soft Computing*, vol. 20, no. 1, pp. 343-357, 2016. Article (CrossRef Link)
[26] Q. D. Ngo, H. T. Nguyen, V. H. Le, and D. H. Nguyen, "A survey of IoT malware and detection methods based on static features," *ICT Express*, vol. 6, no. 4, p. 280-286, 2020. Article (CrossRef Link)
[27] S. Joshua, and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proc. of 2015 10th international conference on malicious and unwanted software (MALWARE)*, IEEE, pp. 11-20, 2015. Article (CrossRef Link)
[28] D. J. Jeon, and D. G. Park, "Real-time malware detection method using machine learning," *The Journal of Korean Institute of Information Technology*, vol. 16, no. 3, pp. 101-113, Mar. 2018. Article (CrossRef Link)
[29] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Proc. of Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. Article (CrossRef Link)
[30] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639-668, 2011. Article (CrossRef Link)
[31] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proc. of the 8th international symposium on visualization for cyber security*, pp. 1-7, 2011. Article (CrossRef Link)

Jaewoong Moon is a professor at Sejong University and has been an expert in information protection and software for 25 years. He majored in information protection in 2019 at Sejong Cyber University Graduate School and completed his doctorate in the Department of Information Protection at Sejong University in 2021. Starting as an engineer, he conducted security consulting, research and development, and system design in the field of cybersecurity, and founded an information protection company in 2002 and operated the company as a CEO for about 16 years.

Subin Kim is a B.S. candidate in the department of computer and information security with the college of software convergence, Sejong University, Seoul, Korea. Her research interests include information security, deep learning, and artificial intelligence.

JangYong Park is starting to work toward the Ph.D. degree in computer and information security at Sejong University, Seoul, South Korea in 2017. He is currently working at the Software Engineering Security Group (SESLab) in Security at Sejong University, and his main research interests are computer security, information security industry development trend research, and air gap.

Jieun Lee received the B.Sc. degree in embedded systems engineering at Daegu University, Daegu, South Korea in 2017, respectively. She is currently working toward the Ph.D. degree in computer and information security at Sejong University, Seoul, South Korea. Prior to her current position, she was a researcher at Autonomous IoT Research Center, Korea Electronics Technology Institute (KETI), South Korea. Her research interests include Internet of Things interoperability, semantics, and data platform testing. Contact her at love9ly@sju.ac.kr.

Kyungshin Kim is a Principal Researcher, leading the Computer Security Development Team in the Agency for Defense Development of Korea. He received a PhD at Department of Computer Engineering of Kyunghee Univ in Korea, and holds B.S and M.S in the Electronic Engineering of Kumoh Institute of Technology and in Electronic Engineering from Yonsei University. Now he focuses on Machine Learning for Malware Detection and Security information and event management (SIEM) in Network Security Monitoring Systems.

JaeSeung Song is an associate professor, leading the Software Engineering and Security Lab (SESLab), in the Computer and Information Security Department at Sejong University. He received a PhD at Imperial College London in the Department of Computing, United Kingdom. He holds B.S. and M.S. in Computer Science from Sogang University. His research interests span the areas of software engineering, software testing, networked systems and security, with a focus on the design and engineering of reliable IoT/M2M platforms, particularly in the context of semantic IoT data interoperability, secure software patch techniques, blockchain IoT, and edge computing.