Journal of Innovations in Computer Science and Engineering
Shahid Beheshti Unversity

June 2025, Volume 3, Issue 1

Malfustection: Obfuscated Malware Detection and Malware Classification with Data Shortage by Combining Semi-Supervised and Contrastive Learning

Mohammad Mahdi Maghouli, *Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran, m.maghouli@mail.sbu.ac.ir*
Mohamadreza Fereydooni, *Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran, m.fereydooni@mail.sbu.ac.ir*
Mojtaba Vahidi-Asl, *Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran, mo_vahidi@sbu.ac.ir*
Monireh Abdoos✉, *Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran, m_abdoos@sbu.ac.ir*

Abstract

With the advent of new technologies, the use of various digital gadgets in different formats is becoming increasingly widespread. In today's world, where everyday tasks are often made easier by technology, the extensive use of computers creates opportunities for malicious activity. Malware is one of the well-known and widely used means by which malicious attackers conduct destructive activities. Producing malware from scratch is challenging, so attackers often obfuscate existing malware and modify it to make it unrecognizable. Since creating new malware from an old one using obfuscation is a creative task, there are some drawbacks to identifying obfuscated malware. In this research, we propose a solution to overcome this problem by converting the code to an image in the first step and then using a semi-supervised approach combined with contrastive learning. In this case, an obfuscation in the malware bytecode corresponds to an augmentation in the image. Hence, by utilizing meaningful augmentations that simulate obfuscation changes and combine them to generate complex ambiguity procedures, our proposed solution can construct, learn, and detect a wide range of obfuscations. This work addresses two issues: (1) malware classification despite data deficiency, and (2) obfuscated malware detection by training on non-obfuscated malware. According to the results, the proposed method effectively addresses the data shortage problem in malware classification, achieving an accuracy of 90.1% when only 10% of the data is used for training the model. Moreover, training on basic malware without obfuscation achieved 96.21 percent accuracy in detecting obfuscated malware.

KEYWORDS

Malware Classification, obfuscated malware detection, semi-supervised learning, contrastive learning, code to image transformation.

1. INTRODUCTION

Today, the increasing number of malware is one of the most significant challenges to information security and communication networks. Automatic malware detection is crucial for users to identify and mitigate the malicious effects of malware. Machine learning methods have become promising and practical techniques for malware detection [38]. The performance of these methods depends on the amount of data collected for analysis. Although everyone has access to vast volumes of data over the internet, data collection and labeling are usually expensive and challenging in most applications. As a solution to this problem, semi-supervised learning methods can be effective in addressing the shortage of labeled data.

Malware detection methods generally fall into three categories: static analysis, dynamic analysis, and hybrid analysis [45]. Statistical analysis detects malware by analyzing the program's assembly code, converted from its binary file. This technique analyzes Portable Executable (PE) [1] files without running them, utilizing the code's opcodes, control flow graph, function call frequency, and system-level APIs.

This method is affected by malware obfuscation due to the use of features such as Control Flow Graphs or the frequency of calls. On the other hand, converting executable code to assembly code and the uncertainty associated with this conversion increase the complexity of this method [2].

Submit Date: 2024-03-11
Revise Date: 2025-07-21
Accept Date: 2025-08-17
✉ Corresponding author

Dynamic analysis attempts to identify malicious behavior by running the program in a controlled environment, such as a virtual machine or sandbox [42]. This method is time-consuming and requires hardware resources. It may limit the program's behavior in a controlled execution environment due to the lack of access. However, the environment cannot fully track the program's behavior. Despite these limitations, dynamic analysis attempts to detect malware by analyzing a sequence of API calls and system calls, classifying them into common categories, such as Ransomware, Worms, etc.

From the binary executable file, the statistical analysis attempts to find similar bit sequences. By using this method, the program is not disassembled or run. Furthermore, since malicious files often employ the same logic or, in some cases, reuse code or utilize malicious libraries, the code can be classified as malware or benign.

The primary challenge of malware detection is related to obfuscated malware, which is created by obfuscating the binary code of a primary malware or another obfuscated malware. As a result of this obfuscation, the primary function of malware remains unchanged, but its appearance changes. Therefore, it is less likely to be detected.

Code obfuscation techniques include disassembling and reassembling, repackaging, data encoding, and code reordering [3, 4]. The more complex the obfuscation, the harder it becomes for antivirus programs that perform static scans to detect malware.

Another point is that obfuscated malware cannot be categorized in several specific ways, as this process depends on the creativity and innovation of the attackers, such that the appearance of the code changes, but its logic and primary purpose remain the same. Code obfuscation is analogous to image noise. Therefore, if we can disregard the obfuscation of the code, which does not alter the identity and nature of the program, we can always distinguish the malicious or non-malicious core of the software. Utilizing a method to detect and eliminate this noise can lead to higher accuracy in malware detection.

The primary challenge posed by obfuscated malware for antivirus programs is that each malware code can generate thousands of obfuscated codes, and each obfuscated code can generate other obfuscated ones similarly. In this regard, in the next generations, the obtained code is significantly different from the original code, making it impossible for antiviruses to identify new generations just by recognizing the primary ones and saving them in their databases.

To accomplish this, we need a method for learning the basic features of malicious code, regardless of how ambiguous the code may be. Moreover, it can also detect a large number of obfuscated codes generated from the main malware by using only a small portion of the original malware code. By achieving a higher level of abstraction in the implemented code, we can identify what the code is, just as we overlook specific details of a landscape with a general vision. Computer Vision methods have matured dramatically over the last few years, thanks to the help of Deep Neural Networks, and have become indispensable for classification tasks. This is why our fundamental idea is to present codes in the image sphere and utilize these new methods. By applying an extensive array of image augmentations and learning using a particular loss function, contrastive learning can identify the core concept of the samples and discard the details that do not contribute to the image's identity. We can utilize this method to detect the malicious or non-malicious core of the software. Moreover, contrastive learning, introduced in SimCLR [11] as a semi-supervised method, is utilized for malware classification and obfuscated malware detection in this paper.

This paper is organized as follows: Section 2 describes the basic concepts. The literature review and the proposed method are presented in Sections 3 and 4, respectively. The experimental results are presented in Section 5, covering malware detection and obfuscated malware detection. Section 6 concludes the paper.

2. BASIC CONCEPTS

This paper proposes a semi-supervised malware classification and obfuscated malware detection method using contrastive learning on code images. The following sections describe semi-supervised learning and contrastive learning as fundamental concepts in machine learning.

2.1. SEMI-SUPERVISED LEARNING

Semi-supervised learning methods are a class of machine learning techniques that utilize both unlabeled and labeled data simultaneously for data classification. Unlabeled data are usually available for most problems; however, providing labeled samples is typically expensive. Therefore, semi-supervised methods have become a promising approach to this problem.

Semi-supervised learning methods have various implementations and utilization solutions. [6]. One of the most prominent methods is to train a model with a small fraction of labeled data and then utilize vast amounts of unlabeled data to generate pseudo-labels for every input using the trained network. To define the loss function, we can consider the loss between the pseudo-label and the predicted probability obtained by passing the data through the network and fine-tuning it [7].

Semi-supervised learning can leverage self-supervised pre-training on unlabeled data using augmentations. [8]. In this regard, we require a suitable loss function to classify the most similar but unlabeled inputs into the same group. Therefore, we utilize a contrastive loss function (for more details, see Section 2-2). The network will be fine-tuned using a small subset of labeled data. As a result, through a semi-supervised approach, we can train the model's encoder (like CNNs) and projection head in an unsupervised way, then replace some projection head layers with new ones and fine-tune the projection head by using a small fraction of data that has class labels in a supervised way [5, 9].

2.2. CONTRASTIVE LEARNING

In many cases, learning data representation is advantageous, especially if gathering labeled data is costly and we have a large amount of unlabeled data. A solution that can help learn representations and features from the data, and contrastive learning, could be beneficial in these cases. Contrastive learning is effective in various applications, including image classification [10], image enhancement [11], feature extraction on time-series data [12], and natural language processing [13, 14].

In contrastive learning, the most important part is defining the contrastive loss function, which guides the model training in a way that maximizes the difference between diverse input data and minimizes the distance between similar data. Contrastive learning has several applications across various domains, some of which are discussed in [15].

A framework for contrastive learning of visual representations, called SimCLR, has been introduced in [16]. This framework employs a semi-supervised learning approach, utilizing image augmentation for automated labeling of images. When an image is converted to another one, the core concept of the image remains unchanged, and therefore, both images should have the same label when contrastive learning has been used. An encoder network transforms each image into a feature vector.. This framework utilizes ResNet [17] as an encoder [16], as illustrated in Figure 1, which takes an image and generates a 1-hot feature vector as its output, known as an "encoder unit". The size of the feature vector is 2048, indicated by h in Figure 1. These feature vectors are then fed as input to another fully connected neural network, which employs a non-linear function [16].

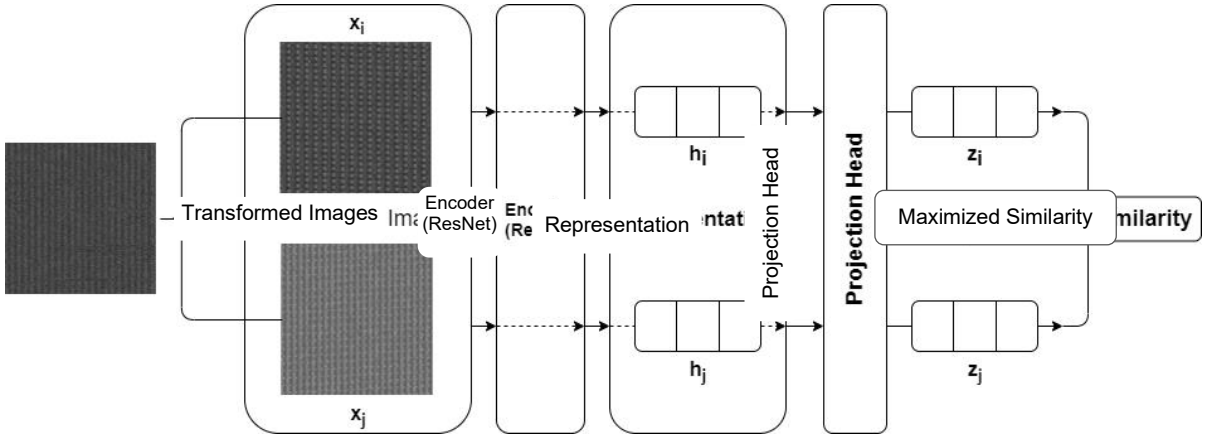


Figure 1. The SimCLR framework [18]

The learning steps are performed in the Projection Head in such a way that the weights of this network are adjusted according to the output vectors. If the inputs are generated from the same image, the similarity is high; otherwise, the similarity is low. This part, which is the main idea of contrastive learning, is shown in Figure 2.

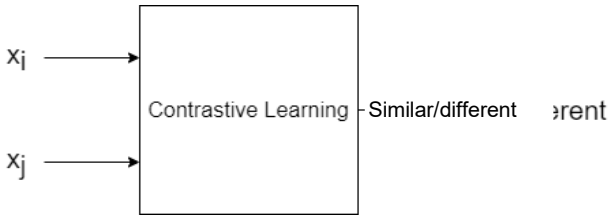


Figure 2. Contrastive learning

It is worth noting that this framework outperforms the state-of-the-art methods for the ImageNet dataset classification. Moreover, this framework can overcome the lack of sufficient images in some applications, as it can generate new data from existing samples. This framework claims that using only 10% of ImageNet data for training can achieve an accuracy of about 80% of the accuracy reached when a large portion of the labeled data is used for training [16].

SimCLR-V2 is an enhanced version of SimCLR based on contrastive learning extended by Google [5]. Network training is performed in three steps within this framework. In the first stage, an unsupervised pre-training with a task-agnostic approach trains an extensive network, regardless of the data labels, in a manner that maximizes data differentiation. Then, it discards half of the projection head and defines a new projection head on top of the trained network. It fine-tunes the trained network using a small amount of labeled data in a supervised manner. To create a lighter model with high accuracy and higher speed, in the third stage, the extensive network obtained is used to train a smaller network, a process known as distillation. The steps of this framework are illustrated in Figure 3, and the pseudo-code of the SimCLR framework is presented in Algorithm 1.

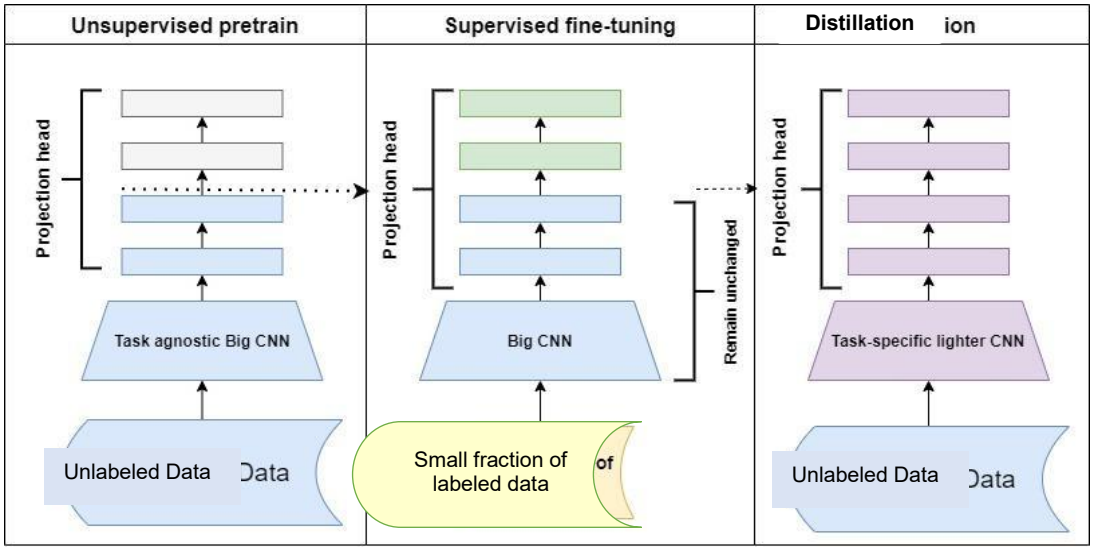


Figure 3. SimCLR-V2 schema [5]

Algorithm 1: SimClr

```
Result: a model to classify instances to predefined classes
create standard TensorFlow dataset with given batch size;
for i ← 1 to epoch_size do
    foreach batch b in AllBatches do
        features =
            make_two_transforms_for_each_data_using_data_augmentation_on_each_image_in_batch();
        hiddenLayersOutput = ResNet(depth, features);
        projectionHeadOutput = model.projection_head(hiddenLayersOutput);
        calculateContrastiveLoss(projectionHeadOutput, b.datas);
        model = tunning_projectionHead_using_constrastiveLoss();
    end
    fine_tunning_fully_connection_layer_using_supervised_data();
end
distilledModel = distilling_model_in_a_task_specific_way(model, dataset);
```

Regardless of the label of the data, the pre-training of the network involves selecting a batch of data, applying two augmentations to each image, and creating two augmented images, as depicted in Figure 4. Consequently, the amount of training data per batch is twice the batch size. Each obtained image is then passed through an encoder, here ResNet101, and its feature vector is extracted.

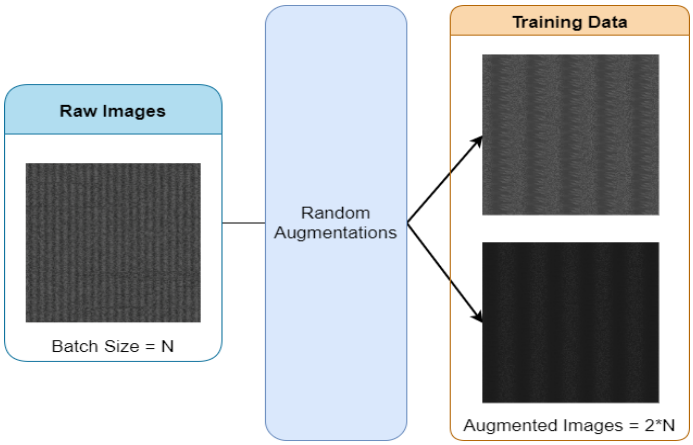


Figure 4. Two augmented samples

Finally, the feature vectors pass through the projection head (fully connected layers), and the final vectors are achieved. Then, in the backpropagation step, a new loss function is defined. The function works in such a way that it tries to bring the image derived from one basic image as close as possible to each other, and the data derived from different photos as far away as possible. The loss function is defined as Eq. (1).

$$l(i,j) = -log \frac{exp\,(s_{i,j})}{\sum_{k=1}^{2N} l_{[k \neq i]} exp\,(s_{i,k})} \qquad (1)$$

In Eq. (1), l(i, j) is the estimated loss, and $s_{i,j}$ is the cosine similarity of two instances derived from one image, and $s_{j,k}$ is the cosine similarity of two instances derived from different images [5]. More details on the contrastive loss function can be found in [19].

The main features of this method are:

- Many augmentations are used for training the network.
- High accuracy can be achieved by training only on a small percentage of labeled data.
- Augmentation can enhance the understanding of the model from the malware core of the code and overcome the challenge of code obfuscation. Another challenge in detecting obfuscated malware is the significantly larger volume of obfuscated malware compared to its underlying malware. On the other hand, SimCLR, by defining a new contrastive loss function, offers a semi-supervised method that addresses the challenges of data scarcity and the high cost of collecting large, labeled datasets. This method can create a high-precision model with only a small percentage of labeled data and a large amount of unlabeled data.

3. LITERATURE REVIEW

In this section, the most important works that examine the malware classification problem or malware detection from either an obfuscated or unobfuscated perspective are reviewed to address the role of obfuscated malware utilizing a deep-learning or machine learning approach.

A method for malware classification was introduced by Verma et al. in [20]. They convert malware code into an image and then use first-order and second-order statistical equations to extract features based on the distribution and correlation of different points in a one-channel image. Malimg [21] is a suitable dataset in this regard, which is used in this work, in addition to the dataset gathered from [22]. By manually extracting features and analyzing them statistically, the codes are classified into different malware classes. The results were 98.04% for precision, 98.06% and 98.05% for recall and the F1 measure, respectively. However, they did not use any obfuscated malware. Since the features have not been extracted automatically, this leads to the loss of a large number of discriminative features. The authors extended their work on malware detection to focus on reducing false-negative errors. Some new statistical features have been used to improve the accuracy on a dataset of 10,000 data points, without any obfuscated data [23].

The effect of obfuscation on the accuracy of the machine learning method studied in [2], which utilizes both static and dynamic malware detection. Although several well-known obfuscation methods have been introduced, they have shown that obfuscation has a greater effect on the accuracy of static methods than on dynamic methods [2].

Convolutional Neural Networks (CNNs) have been utilized for malware classification on code that is converted into three-channel images in [24]. New images are generated using different types of noise as augmentation. The results showed 96% accuracy on a dataset gathered from GitHub. In addition to benign data, malware is categorized into various classes. However, obfuscated malware has not been explicitly used in their method, which makes it much more challenging to detect [24]. While traditional CNN-based approaches remain popular, transformer-based architectures [39] have shown superior performance in recent studies.

Obfuscation has been employed to address the issue of data deficiency in [25]. The authors attempted to increase the amount of data by mapping a generated code to an image and applying augmentation to the image. Then, transfer learning has been used on a pre-trained network for malware classification. They achieved 93.8% accuracy on the Microsoft 2015 Dataset [26] and 98.5% accuracy on the Malimg Dataset.

Obfuscated malware detection has also been studied in [27], with a focus on Android applications. It has been shown that some malicious Android applications are re-released by obfuscating the package name and certificate owner name. Therefore, the correlation between these two parameters has been used for the classification. Stacking techniques are used to combine Recurrent Neural Networks (RNNs) and CNNs to determine whether an application is malicious or not based on information acquired from the package name and the certificate owner. It is clear that since the features depend on the operating systems, they cannot be extended to other operating systems such as Linux or Windows.

Miller et al. [28] presented a method for malware detection on Android. Four standard obfuscation methods are employed: the presence of a selection of API calls and Android commands, permissions, and opcode instructions. The Discriminative Adversarial Network (DAN) is used to train the network to distinguish between obfuscated malware and benign samples. The main limitation of this method is that it trains the obfuscation while there are different types of obfuscation by attackers, which are not predictable for the network [28].

In recent years, obfuscated malware detection has been considered in various kinds of literature. This type of malware, derived from other malware, generates a vast amount of malware every day, which is widely used for various purposes, such as ransomware. It is crucial to detect them as metamorphic malware can change from one machine to another. Generative models are widely used for detecting obfuscated malware. After training the network, the probability distribution of the occurrence of any point is estimated by a generative network, and obfuscated malware is detected using this estimated value. Due to the creative nature of obfuscation, generative networks cannot consider all possible obfuscation methods, and therefore, the goal of detecting obfuscated malware cannot be fully achieved.

Chen et al. [29] showed that minor changes to the corresponding images could reduce the performance of deep neural networks. On the other hand, there are some solutions based on generative networks that can be applied to counter these types of attacks

[30]. The vulnerability of deep neural networks to adversarial changes has been addressed in [31]. While discussing the types of attacks, it has been demonstrated that malware can bypass detection using this approach.

In [32], an ensemble method called MalNet was introduced for malware detection, achieving 99.36% accuracy on the Microsoft 2015 dataset [26]. MalNet converts binary codes and OPcodes to images and then uses CNN and LSTM for the classification.

A hybrid model composed of ResNet and GoogleNet has been utilized for malware detection in [33], where byte codes are directly converted into images for training. The model achieved 88% accuracy.

Daram et al. proposed an ensemble method for malware classification [34]. Two models were ensembled for the classification of malware on the Microsoft 2015 dataset [26]: in the first one, static features such as file size and n-gram opcodes were extracted and used for the XGBoost model; in the second one, the executable files were converted to images, and the features extracted by a CNN. Since the dataset does not include benign data, it cannot be used as a practical solution for malware and benign detection. A summary of related works is given in Table 1.

Recent advancements by Gao et al. [44] have further improved the SimCLR framework specifically for cybersecurity applications, achieving better performance with fewer labeled samples.

Table 1. Summary of related works

Authors	Approach	Method	Accuracy	Dataset
N. Marastoni et al. 2021 [25]	Malware classification	LSTM + CNN	93.8 %	Microsoft 2015
			98.5%	Malimg
A. Deram et al. (2021). [34]	Malware classification	Ensemble	99.12%	Microsoft 2015
Khan et al. 2018 [33]	Malware classification	ResNet 101	78.84%	Microsoft 2015
		ResNet 152	88.36%	
Yan et al. 2018 [32]	Malware / benign detection	MalNet	99.13%	Microsoft 2015 + collected benign samples
F. Catak et al. 2020 [24]	Malware classification	CNN	96%	Mal-API-2019
V. Verma et al. 2020 [23]	Malware / benign detection	Texture statistics	99.61%	Custom dataset
W. Lee et al. 2019 [27]	Malware classification	CNN-RNN-PA	99.86%	VirusTotal
V. Verma et al. 2020 [20]	Malware classification	CNN	98.58%	Malimg

While our approach leverages well-established frameworks such as SimCLR and ResNet, its main novelty lies in the novel application of contrastive learning to the domain of binary code classification through image-based representations. Specifically, we introduce a tailored data preprocessing strategy that transforms binary code into visual formats suitable for deep contrastive learning, enabling effective feature extraction and classification. Additionally, our method demonstrates significant improvements in detection performance metrics compared to traditional binary analysis approaches, highlighting its practical value and contribution to advancing malware detection techniques.

4. THE PROPOSED METHOD

As mentioned earlier, to overcome the challenge of detecting obfuscated malware, it is essential to disregard the noise and focus on the malicious core of the code. For this purpose, we use a semi-supervised learning method described in Section 2-1. Our approach addresses advanced obfuscation techniques that resemble adversarial attacks [40], ensuring robust detection capabilities. In this regard, the code should be mapped to an image to create a higher-level abstraction of the binary codes, i.e., minor changes do not significantly affect the overall view of the image. Therefore, the code-to-image conversion implicitly overcomes minor obfuscations. Image augmentation is typically used to enhance the identity of images in semi-supervised methods. Augmentation also helps generate new samples from an image to address the data scarcity problem. Each method of augmentation can be considered as an obfuscation approach. For example, a vertical flip corresponds to the obfuscation of malware, which is created by rearranging code parts or reordering them in a way that does not affect the main program's logic. Even a few degrees of blur can also be considered as changing some details without altering the general concept of the image, which could be analogous to adding or removing unnecessary lines of code from the malware. The creation of new augmented samples is generally performed, particularly for classes where collecting data is challenging. Alongside a semi-supervised method, we use a contrastive learning method for data clustering. As explained in Section 2, this method categorizes samples by differentiating and contrasting instances of different classes, while maximizing the similarity between instances of the same class. As a result, the proposed method combines contrastive learning and semi-supervised learning, generating new samples through augmentation and classifying them as described.

The general framework of our proposed method is shown in Figure 5. First, two sets of benign ware and malware are collected. Benign ware data is collected from Windows system files and cleaned so that it resembles malware format. The binary codes of the Windows operating system are selected because they contain API calls and low-level system calls, as well as high-level permissions, which make them difficult for malware to detect.

Malware data is collected from the Microsoft 2015 dataset [26], which is presented in nine classes, as detailed in Section 5. One of them is called Obfuscator.ACY contains only obfuscated malware, and the other classes include primary malware in each class (malware with no obfuscation). This data is then converted to grayscale images so that every 8 bits corresponds to a value

of intensity in pixels. Images are generated at a fixed width of 1024 pixels and a variable length depending on the size of the binary file. However, the images are resized and randomly cropped in augmentation at the beginning of the process.

In this research, two main problems have been addressed to show the efficiency of the proposed method. The first problem is malware classification, and the second is detecting obfuscated malware from benign programs, known as obfuscated malware detection. The solution to the first problem focuses on the lack of data, as training was conducted with only 20% and 10% of the data, and testing was performed on the remaining 80% and 90%, respectively, as described in the next section.

4.1. CONVERTING CODES TO IMAGES

In this method, the software's bytecode is converted into an image. The bytecode is reshaped into a binary image as described in [30], i.e., a two-dimensional array with a size of 1024 × h, where h depends on the length of the code. Figure 6 illustrates an executable bytecode sample, and Figure 7 shows an example of an image obtained from a bytecode as described in Algorithm 2.

4.2. MALWARE CLASSIFICATION

First, an extensive network is trained with a task-agnostic approach in an unsupervised way. In this regard, in each iteration, a batch of images is selected, and two augmentation methods are applied to each image. These augmented images are then passed through the encoder network for feature extraction. As shown in the figure, ResNet-101 is used as the encoder.

According to the results reported in [5], when a problem involves a data shortage, utilizing deeper networks along with larger batch sizes can lead to a better outcome. Therefore, ResNet200 is used to train the network with 20% of the data and to evaluate the remaining 80% in the first experiment, and with 10% of the data for training and 90% for evaluation in the second experiment, as depicted in Figure 8. The generated feature vectors from the encoder unit then pass through the projection head, where the projection head nodes are tuned during the training phase. In contrastive learning, using the loss function as defined in Eq. (1), errors between similar images are minimized while errors between different images are maximized. In the next stage, a part of the obtained projection head is discarded and replaced by a new projection head. Using a small part of the data, the network is fine-tuned in a supervised way. Therefore, in a task-specific approach, a smaller network is trained with a larger network to produce a lighter, faster, and reasonably accurate model.

4.1. OBFUSCATED MALWARE DETECTION

To detect obfuscated malware, as shown in Figure 5, the data is prepared in the dataset preparation module, and then the data is divided into two parts using a train/test splitter:

1. The train set, which contains 80% of the whole benignware and all the basic malware (without any obfuscation)
2. The test set, which contains the remaining 20% of benign wares and the whole Obfuscator.ACY class in the Microsoft 2015 dataset, which contains only obfuscated malware, as illustrated in Figure 9.

For malware data, all the data in the Microsoft 2015 dataset is used for training the network, except for the Obfuscator.ACY, which is used for the test phase. Therefore, the model is tested with a category of data that has never been seen before, namely, obfuscated malware. We demonstrate that we can predict many obfuscated malware samples from benign code through our experiments. The obfuscation in code can be seen as a form of augmentation in an image, allowing our network to learn how to ignore obfuscation in corresponding images of code and determine whether the code is malicious or not. For generalization, the SimCLR framework is used to generate augmented images from the base images in the training set. In contrastive learning, the purpose of the training method is to reduce the difference between the augmented images from the same source. In this regard, the network learns to ignore augmentation in images, which acts as obfuscation in the corresponding code. According to Figure 4, the SimCLR framework generates two augmented images from each image in the batch during the training module. To accomplish this, we first resize the shorter side of the image to 224 pixels and resize the other side proportionally to preserve the image's aspect ratio. Then, 224×224 random crops are generated, and an augmentation is randomly selected and applied. As a result, two output images with different combinations of augmentations are generated from each input image. The desired augmentation includes random flip, color distortions, reshaping, and blurring. Figure 10 shows an example of augmentation.

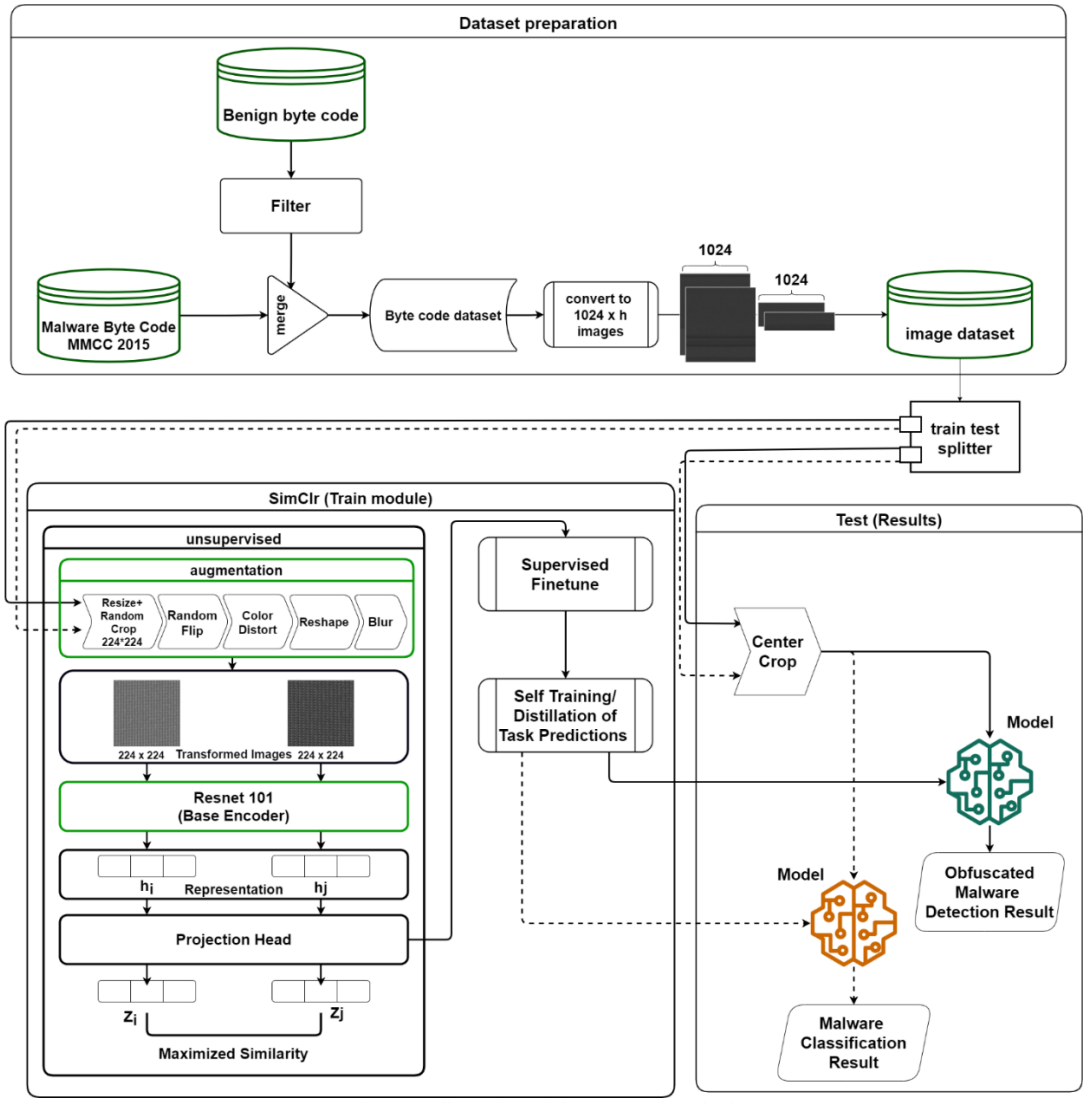


Figure 5. Illustration of the proposed method

```
00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
00401010 BB 42 00 8B C6 5E C2 04 00 CC CC CC CC CC CC CC
00401020 C7 01 08 BB 42 00 E9 26 1C 00 00 CC CC CC CC CC
00401030 56 8B F1 C7 06 08 BB 42 00 E8 13 1C 00 00 F6 44
...
```

Figure 6. A malware bytecode sample

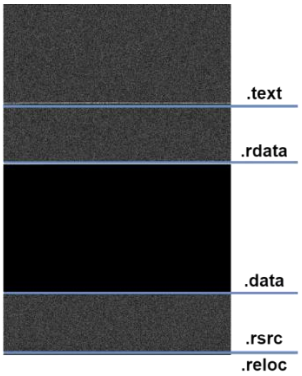


Figure 7. Result of converting a code to an image [35]

Algorithm 2: Binary Code to Image Convertor

```
Result: 1024 × h PNG images
prepare binary code files in dataset;
foreach file f in bytecodesDataset do
    bytes = f.readBytes();
    width = 1024;
    height = len(bytes)/width;
    image = bytes.reshape(width, height);
    saveBytesAsPNG(image);
end
```

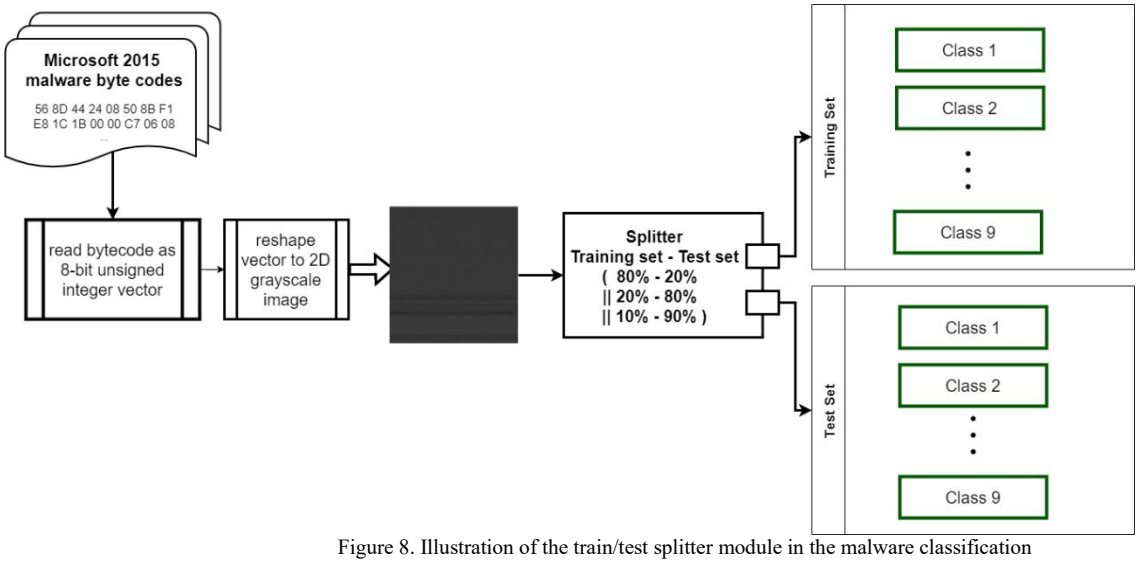


Figure 8. Illustration of the train/test splitter module in the malware classification

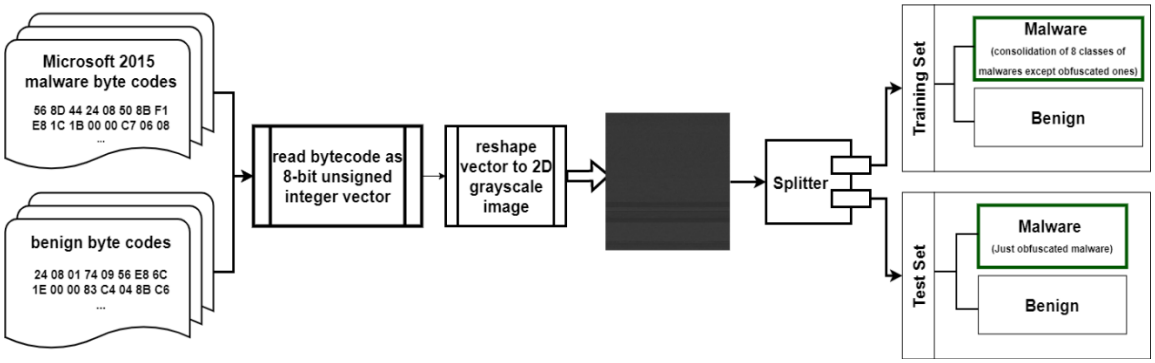
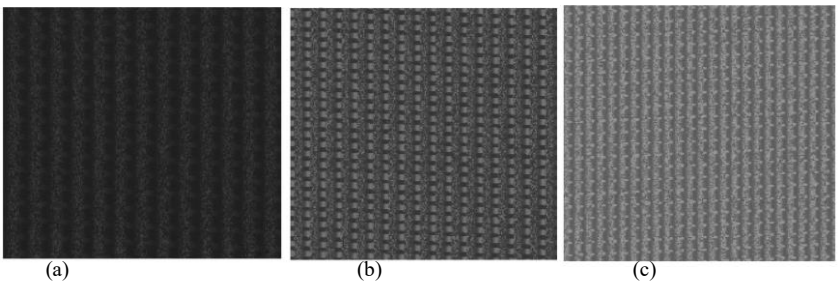


Figure 9. Illustration of the train/test splitter module in the obfuscated malware detection problem



(a) (b) (c)

Figure 10. a) Transferred image from source code; b) Code image after random crop and color distortion; c) Code image after performing random crop and blur.

The augmented images are then used, and the corresponding original images are discarded. These images are given to an encoder to create feature vectors from the images. ResNet101 is used for extracting a 2048-dimensional feature vector. The feature vectors are fed into a fully connected network in the projection head to train the network's weights. The trained network is tested on a dataset that includes obfuscated malware and benign software. Data preprocessing is applied to test images by resizing them to 250 pixels on the shorter edge, while maintaining the ratio, and then cropping them to 224×224 as part of the test phase.

Analysis- Converting code into images allows leveraging the power of deep learning models designed for visual data, such as CNNs. This transformation simplifies complex code structures into recognizable visual patterns, enabling more effective feature extraction, improving classification accuracy, and benefiting from transfer learning with pre-trained models. While analyzing binary code directly is possible, the visual approach offers a more efficient and robust framework, particularly for tasks like malware detection and code analysis [46]. Recent work on explainable AI by Nguyen et al. [43] demonstrates that image-based representations can provide more interpretable detection results compared to traditional binary analysis methods, while maintaining high accuracy.

5. EXPERIMENTAL RESULTS

In this section, the dataset used in the experimental results and evaluation metrics is described. The results are presented in two parts: the findings on malware detection and the results for detecting obfuscated malware.

5.1. DATASET

The Microsoft Malware Classification Challenge dataset (Microsoft 2015) [26] has been used to evaluate the proposed methods. This dataset includes the bytecode of malware classified into nine malware classes, used for a contest held by Microsoft on Kaggle in 2015 [36]. The classes are not balanced in terms of data distribution, making the classification task more challenging. The number of samples in each class varies from 42 to 2942 samples. This dataset has been widely used in [25, 32-34] and has also been used to address challenges. The details of the samples in the dataset are given in Table 2.

Table 2. Malware classes in Microsoft 2015 Dataset [26]

Class Name	Number of training samples	Type
Ramnit	1541	Worm
Lolipop	2478	Adware
Kelihos_ver3	2942	Backdoor
Vundo	475	Trojan
Simda	42	Backdoor
Tracur	751	TrojanDownloader
Kelihos_ver1	398	Backdoor
Obfuscator. ACY	1228	Any obfuscated malware
Gatak	1013	Backdoor

Our training set for obfuscated malware detection includes samples from all classes except the Obfuscator class.ACY, which is used only for the test set. In this way, the network is trained using malware samples without obfuscation to learn the primary features of each sample.

Additionally, due to the lack of proper, standardized benign datasets, Windows system files are used as benign codes because they invoke low-level system APIs, just as malware does. Therefore, they are more difficult to distinguish from malware, which can be used to evaluate the proposed method. For this purpose, the data are first collected and then cleaned in a manner that ensures their structure is similar to that of the data in the malware class. To preserve the structure of these files similar to the Microsoft 2015 dataset, only 32-bit programs were used, and only the main parts of the file were preserved. Also, the binary code related to PE Headers was removed from the beginning of the files. These processes were performed using the "pefile" library [37].

After code preparation, each bytecode is mapped to the brightness of a pixel, resulting in a 1-channel grayscale image with a fixed width of 1024 pixels and a variable height depending on the code size. Using one-channel images has the advantage of allowing for analysis with less data.

5.2. EVALUATION METRICS

Eqs (2)-(7) are applied to each algorithm as performance measures to achieve an efficiency comparison of the proposed model. T, F, P, and N stand for True, False, Positive, and Negative, respectively. In Eq. (7), β is a positive real factor.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2)$$

$$TPR = \frac{TP}{TP + FN} \quad (3)$$

$$FPR = \frac{FP}{FP + TN} \quad (4)$$

$$Precision = \frac{TP}{TP + FP} \quad (5)$$

$$Recall = \frac{TP}{TP + FN} \quad (6)$$

$$F - Score = \frac{1 + \beta^2 * Recall * Precision}{\beta^2 * (Recall + Precision)} \quad (7)$$

5.3. RESULTS ON MALWARE DETECTION

As mentioned earlier, the proposed method is being evaluated using the Microsoft 2015 dataset [14], which includes nine different classes of malware. To demonstrate the performance of the proposed method, three experiments were conducted. In the first experiment, 80% of the samples from each class were included in the training set, while the remaining 20% were used as test data. In the second experiment, 20% of the samples were used for training, and the remaining 80% were used as a test set. In the third experiment, 10% and 90% of the samples were used in the training set and test set, respectively. The number of samples used for training and testing the network is given in Table 3. The hyperparameter values used in the training phase of the three experiments are presented in Table 4. For resource-constrained environments, recent work by Sharma et al. [47] shows that knowledge distillation techniques can effectively reduce model size while maintaining detection accuracy, a direction worth exploring in future implementations.

5.3.1. THE FIRST EXPERIMENT

The first experiment performed network training using 80% of the data, while the other 20% used for the test phase. The values of the hyperparameters used in this experiment are reported in Table 4. The accuracy and loss values during the training phase are shown in Figure 11.

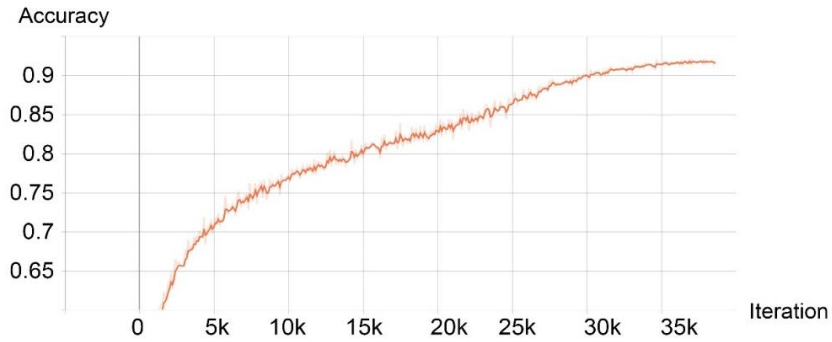
The accuracy of the model is 94.11% in the first experiment, while the accuracies achieved by transfer learning and LSTM [20] are 90.8% and 93.8%, respectively.

Table 3. Distribution of data for each class of malware in each experiment

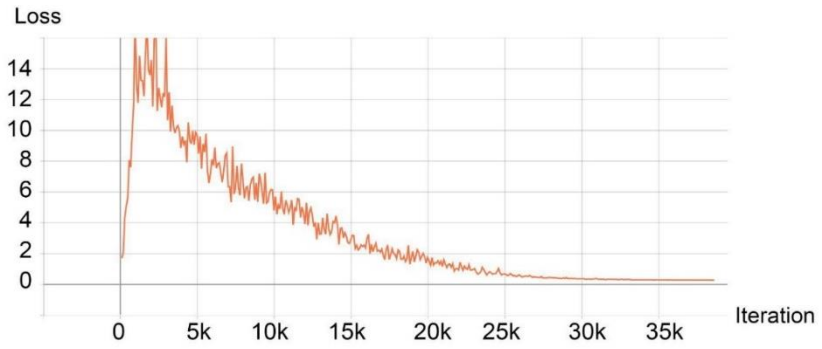
Malware class	experiment	Training samples	Test samples
Ramnit	1st experiment	613	148
	2nd experiment	148	613
	3rd experiment	77	684
Lollipop	1st experiment	1895	471
	2nd experiment	471	1895
	3rd experiment	281	2085
kelihos_ver3	1st experiment	2354	588
	2nd experiment	588	2354
	3rd experiment	294	2648
vundo	1st experiment	112	34
	2nd experiment	34	112
	3rd experiment	13	133
simda	1st experiment	31	8
	2nd experiment	8	31
	3rd experiment	9	30
Tracur	1st experiment	460	96
	2nd experiment	96	460
	3rd experiment	72	484
Kelihos_ver1	1st experiment	57	18
	2nd experiment	18	57
	3rd experiment	7	68
Obfuscator.ACY	1st experiment	93	17
	2nd experiment	17	93
	3rd experiment	40	70
Getak	1st experiment	554	96
	2nd experiment	96	554
	3rd experiment	172	478

Table 4. The network hyperparameter values in each experiment

	Number of epochs	Batch size	Encoder	Initial learning rate	Temperature	Weight decay	Training data dimensions	Augmentation used?
1st experiment	400	64	ResNet101	1.0	0.5	1e-4	224*224	YES
2nd experiment	1000	32	ResNet200	1.0	0.5	1e-4	224*224	YES
3rd experiment	1000	32	ResNet200	1.0	0.5	1e-4	224*224	YES



(a)



(b)

Figure 11. a) Accuracy during training the model, and b) loss value during the training of the model in the first experiment

In [34], the accuracy using CNN on all data is 98%, while it is 99.12% using an ensemble method. However, the accuracy of our proposed method in the training phase is higher than the method presented in [34]. The training phase has been done on a large portion of the data. The accuracy of the method on the test phase has been reported as 98% using CNN and 99.12% using the ensemble method. Although the accuracy of their method is higher than our proposed method in the test phase, our approach achieves the same accuracy as their method using CNN in training, which demonstrates the effectiveness of our proposed method when a large portion of the data is utilized. The obtained confusion matrix and heat map are presented in Tables 5 and 6.

5.3.2. THE SECOND EXPERIMENT

As described earlier, to demonstrate the effectiveness of the proposed method, a small fraction of labeled data is used for training the model. In the second experiment, 20% of the data is used for training. Therefore, a larger batch size and greater depth of the network are needed to achieve better results. The number of epochs used in this experiment is

1000, while the batch size is 32, and ResNet200 has been used as the encoder. Using 20% of the data for training, the proposed method achieved 91.95% accuracy on the remaining 80% of unseen data. The accuracy and loss values during the training phase in this experiment are shown in Figure 12. Additionally, the confusion matrix and heat map are presented in Tables 7 and 8, respectively.

5.3.3. THE THIRD EXPERIMENT

In the third experiment, 10% of the labeled data was used for training the network. The network parameters are shown in Table 4. The accuracy of the proposed method, using only a small number of labeled data, is 90.1%. It means that by using a small fraction of the data (10%), the proposed method achieves 97.75% of the best accuracy of the method presented in [25], which uses 80% of the data for training, and 92.42% of the accuracy reported in [34] on the entire dataset. This result demonstrates that utilizing contrastive learning and SimCLR yields exceptionally high detection accuracy with a limited amount of data.

Given that a relatively small amount of labeled malware data is available, this experiment demonstrates that the proposed method is effective for malware detection not only when a small dataset is available, but also when a larger dataset is available. Since the number of labeled data used for training is limited, contrastive learning provides a faster and lower-cost method (in terms of data collection) for malware detection. The obtained results are illustrated in Tables 9 and 10. Moreover, the final results for each experiment and metric are presented in Table 11, and a comparison among these three experiments is provided in Table 12.

5.4. RESULTS ON OBFUSCATED MALWARE DETECTION

As described in Section 4-2-1, the bytecode of the malware and the benign files are each directly converted into an image, and then the deep network is trained on the basic malware and the benign using the SimCLR-V2 framework.

The two main problems in obfuscated malware detection are as follows:

1. Excessive obfuscation in the code in such a way that its functionality remains the same, but its appearance is entirely different.
2. A large number of obfuscated malware compared to the basic ones.

A deep convolutional neural network (DCNN) can be effectively used to address these two challenges.

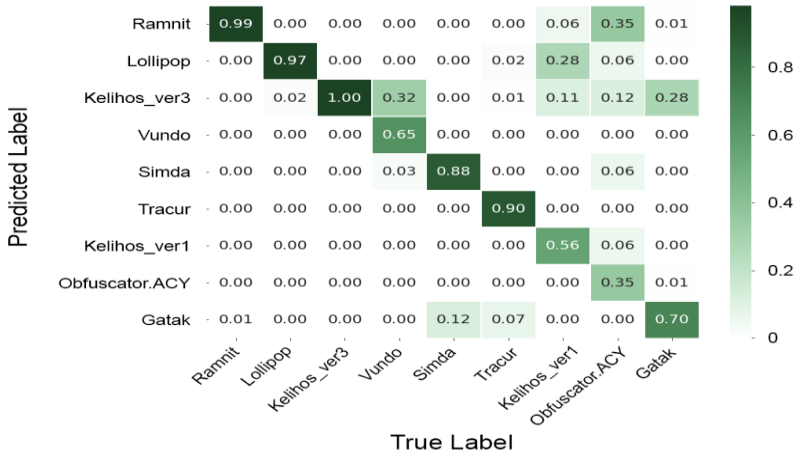
Table 5. Reported confusion matrix for first experiment (columns: True labels, rows: Predicted labels).

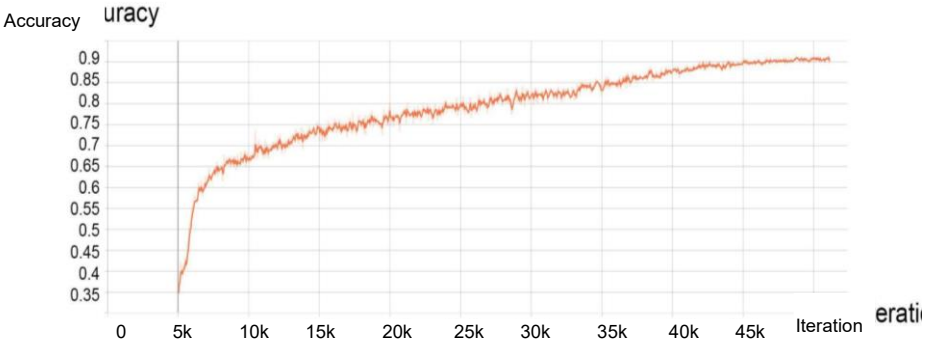
	1	2	3	4	5	6	7	8	9
1	147	1	0	0	0	0	1	6	1
2	0	458	0	0	0	2	5	1	0
3	0	8	586	11	0	1	2	2	27
4	0	0	0	22	0	0	0	0	0
5	0	0	0	1	7	0	0	1	0
6	0	2	0	0	0	86	0	0	0
7	0	0	0	0	0	0	10	1	0
8	0	1	0	0	0	0	0	6	1
9	1	1	2	0	1	7	0	0	67

5.4.1. RESNET WITHOUT AUGMENTATION

In this method, the dataset is divided into three main parts: the training set, the validation set, and the test set. Cross-validation has been employed to evaluate the model. ResNet101 was used in this model on images without augmentation. For training the network, the training data were divided into five equal parts. In each round, four parts were used for training, while the remaining part was used for validation. The images were 224x224 for training, and the best model was selected based on the accuracy achieved during validation. The test data includes unseen benign and obfuscated malware files, which achieved accuracies of 96.27% and 69.18% on the validation and test sets, respectively.

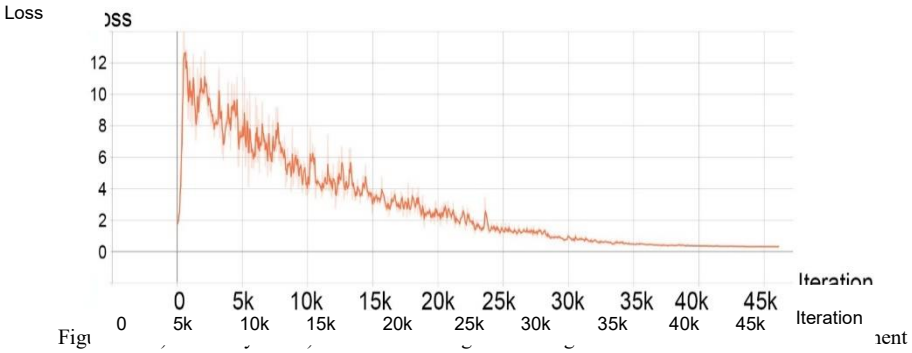
Table 6. Reported heat map for the first experiment, the numbers indicate the recall rate (columns: accurate labels, rows: predicted labels).





(a)

(b)



Figu ıent

Table 7. Reported confusion matrix for the second experiment (columns: True labels, rows: Predicted labels).

	1	2	3	4	5	6	7	8	9
1	575	7	0	0	1	5	3	35	3
2	21	1843	1	0	0	4	11	13	16
3	1	26	2342	65	3	155	7	8	7
4	0	0	0	46	1	1	0	2	0
5	1	1	0	0	23	0	0	0	1
6	3	7	3	1	3	287	1	9	26
7	0	1	2	0	0	1	34	1	0
8	6	0	0	0	0	2	1	21	0
9	5	10	6	0	0	5	0	4	501

Table 8. Reported heat map for the second experiment (the numbers indicate the recall rate): (columns: True labels, rows: Predicted labels).

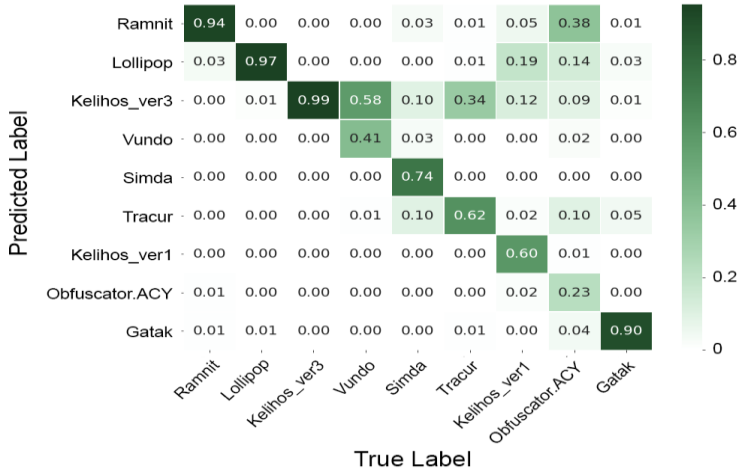


Table 9. Reported confusion matrix for the third experiment (columns: True labels, rows: Predicted labels).

	1	2	3	4	5	6	7	8	9
1	635	7	0	0	0	11	4	28	9
2	15	1990	3	0	0	1	17	3	9
3	2	59	2627	94	2	209	14	9	25
4	0	0	0	36	0	0	0	1	0
5	1	5	0	1	22	7	0	1	1
6	9	14	13	2	4	247	1	4	22
7	1	0	1	0	0	1	32	3	1
8	14	1	3	0	2	5	0	14	2
9	7	9	1	0	0	3	0	7	409
total	684	2085	2648	133	30	484	68	70	478

Table 10. Reported heat map for the third experiment (the numbers indicate the recall rate), (columns: True labels, rows: Predicted labels).

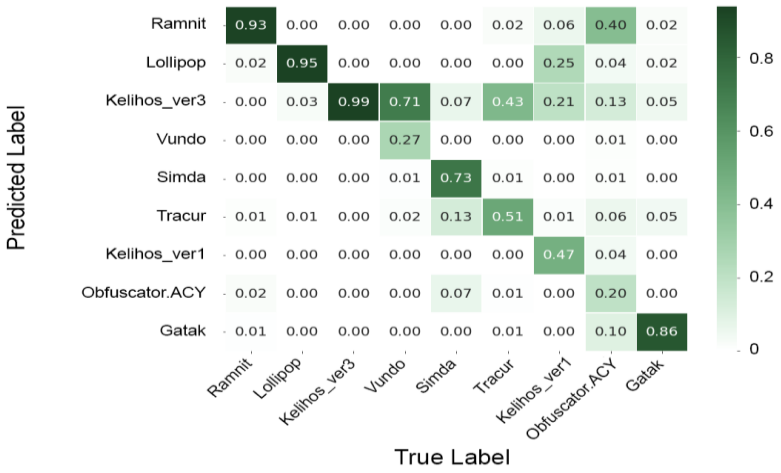


Table 11. Test phase comparative metrics results for all three experiments

Class	Experiment	N(truth)	N (classified)	Accuracy (%)	Precision	Recall	F1-score
Ramnit	1st	148	156	99.32	0.94	0.99	0.97
	2nd	612	629	98.52	0.91	0.94	0.93
	3rd	684	694	98.38	0.91	0.93	0.92
Lollipop	1st	471	466	98.58	0.98	0.97	0.98
	2nd	1895	1909	98.09	0.97	0.97	0.97
	3rd	2085	2038	97.86	0.98	0.95	0.97
Kelihos_ver3	1st	588	637	96.41	0.92	1	0.96
	2nd	2354	2614	95.4	0.90	0.99	0.94
	3rd	2648	3041	93.49	0.86	0.99	0.92
Vundo	1st	34	22	99.19	1	0.65	0.79
	2nd	112	50	98.87	0.92	0.41	0.57
	3rd	133	37	98.53	0.97	0.27	0.42
Simda	1st	8	9	99.8	0.78	0.88	0.82
	2nd	31	26	99.82	0.88	0.74	0.81
	3rd	30	38	99.64	0.58	0.73	0.65
Tracur	1st	96	88	99.19	0.98	0.90	0.93
	2nd	460	340	96.34	0.84	0.62	0.72
	3rd	484	316	95.42	0.78	0.51	0.62
Kelihos_ver1	1st	18	11	99.39	0.91	0.56	0.69
	2nd	57	39	99.55	0.87	0.60	0.71
	3rd	68	39	99.36	0.82	0.47	0.60
Obfuscator.ACY	1st	17	8	99.12	0.75	0.35	0.48
	2nd	93	30	98.69	0.7	0.23	0.34
	3rd	70	41	98.76	0.34	0.20	0.25
Getak	1st	96	79	97.22	0.85	0.70	0.77
	2nd	554	531	98.65	0.94	0.90	0.92
	3rd	478	436	98.56	0.94	0.86	0.89

Table 12. Comparison of our three approaches with other methods

Method	The percentage of data used for training	The percentage of data used for the test	Accuracy
Transfer learning [18]	80%	20%	93.8%
Ensemble method [28]	-	100%	99.12%
SimCLR-V2 (1st experiment)	80%	20%	94.11%
SimCLR-V2 (2nd experiment)	20%	80%	91.95%
SimCLR-V2 (3rd experiment)	10%	90%	90.1%

5.4.2. RESNET WITH AUGMENTATION

This experiment was also performed based on the same data as described in Section 2-1, except that in each iteration, Image augmentations were applied to investigate if they function similarly to code obfuscation. The goal is for the model to gain a better understanding of the code image by utilizing augmentation in order to learn the malicious core of the data.

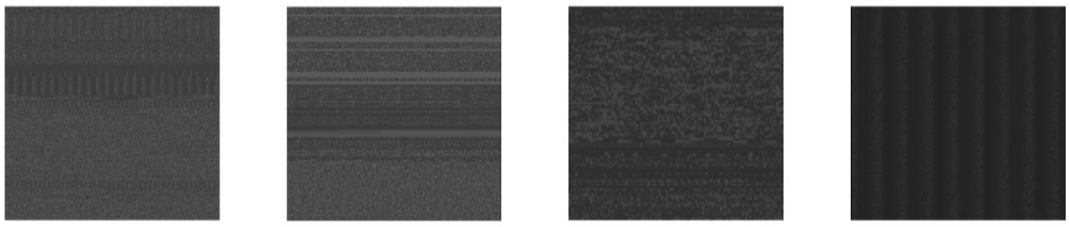
The augmentation techniques applied—such as random resize crop, vertical flip, blur, and color jitter—are motivated by their ability to simulate various obfuscation strategies encountered in malware, including code reordering, redundant code insertion, and appearance modification. These transformations are inspired by the analogy between image noise and code obfuscation: minor modifications that do not impact the core behavior are modeled through these augmentations. For example, vertical flips mimic code rearrangements, while blurring introduces noise akin to redundant or non-essential code. Color jitter and resize cropping simulate changes in appearance that malware might employ to evade static detection. Our experimental results demonstrate that these augmentations significantly improve the model's robustness, enabling it to generalize better against complex obfuscation techniques, as evidenced by the increased detection accuracy in our tests.

In general, standard methods of code obfuscation can be divided into the following categories:

1- Methods that lead to changes in control flow, such as code reordering, splitting functions into two separate functions, and adding switch-case structures for selecting the next basic block in the program's control flow (Known as Flatten), etc.
2- Methods that add or remove redundant codes, such as adding and assigning useless variables, inserting useless functions without calling them, or adding useless conditions or loops that do not change the logic of the code.
3- Methods that modify the appearance of the code while preserving its logic. For example, changing a computational expression in such a way that the output of the new expression is the same as the previous one for similar inputs, but performs different operations.

According to the different obfuscation methods, a set of image augmentations is selected, including random resize crop, vertical flip for the first category, blur, and color jitter for the second and third categories, respectively. Figure 13 illustrates the various types of image augmentation.

- Random resize crop and vertical flip: This augmentation helps to learn different parts of the image separately. In the final model, the displacement of the image elements has no significant effect on the model's output. As a result, it can overcome the problems caused by changes in the control flow.
- Image blurring increases the noise in the image, causing slight changes that do not alter the image's concept. Although some parts of the image may be distorted, it retains the core and concept of the image. This resembles changes from the second category of obfuscation.
- Color jitter includes changing the brightness, contrast, and saturation of the image. Applying these modifications to the image results in a slight change in its appearance, but the image's identity remains unchanged. This kind of augmentation is a good alternative for the third category.



(a) (b) (c) (d)

Figure 13. Different types of augmentations applied to an image: a) Original, b) After color Jitter (including changes in brightness, saturation, and contrast), c) Blurred, d) Random resize cropped

These augmentations are applied randomly to all images in each iteration, and then the model is trained using the new augmented images. The only difference between this approach and the one elaborated in Section 5-4-1 is that the original dimensions of the images are used for training. Since all augmentations apply to the image, a 224x224 random resize crop image is created for network training.

The results show 97% accuracy on the validation set and 89.23% on the test set. It shows that applying augmentation on images helps recognize obfuscated malware from benign code. The results demonstrate a 20% improvement in the accuracy of obfuscated malware detection compared to the previous approach. According to the results, the hypothesis presented in this paper is accurate, and image augmentation helps detect obfuscated malware, proposing a suitable solution to one of the primary challenges in malware detection. Algorithm 3 shows the pseudo-code of the image augmentation methods.

Algorithm 3: Image Augmentation on Batch

```
Result: augmented image
select a batch from dataset;
foreach image im in batchOfImages do
    im = randomSelectionCropWithResize(im, 224, 224);
    p = randomFloat(0,1);
    if p < 0.5 then
        im = randomFlip(im);
    end
    p = randomFloat(0,1);
    if p < 0.8 then
        im = randomColorJitter(im);
    end
    im = reshape(im, channels = [224, 224, 3]);
    p = randomFloat(0,1);
    if p < 0.5 then
        im = randomBlur(im);
    end
end
```

5.4.3. SIMCLR

The same experiments for obfuscated malware detection were performed using the SimCLR framework. ResNet101 has been used for the network encoder. The batch size and number of epochs used for training are 32 and 140, respectively. The test set includes only the obfuscated malware and benign code. The method's accuracy is 98% and 96% on the training set and test set, respectively.

In contrast, the ResNet with the augmentation method, as described in Section 5-4-2, achieved an accuracy of 89.23% in the test evaluation. The results show a significant improvement compared with ResNet101, with an accuracy of 8%. It confirms that applying SimCLR with a higher number of augmentations and utilizing contrastive learning leads to a deep understanding of the malicious core of the code. It can detect obfuscated malware generated from basic malware, as well as ones that will be produced in the future, with an even higher level of abstraction and accuracy. The hyperparameter values used in the experiment are given in Table 13. Additionally, the obtained confusion matrix and heat map are presented in Tables 14 and 15, respectively.

Table 13. Hyperparameters used in the experiments

	#Epoch	Batch Size	Encoder	Initial Learning Rate	Temperature	Weight Decay	Training Data Dimensions	Augmentation Used?
ResNet	240	32	ResNet101	1.0	0.5	1e-4	224*224	NO
ResNet + Augmentation	240	32	ResNet101	1.0	0.5	1e-4	224*224	YES
SimCLR	140	32	ResNet101	1.0	0.5	1e-4	250*250	YES

Table 14. Reported confusion matrix for SimCLR obfuscated malware detection

	malware (true)	benign(true)
malware(predicted)	1187	51
benign (predicted)	41	1148
total	1228	1199

Table 15. Reported heat map for the obfuscated malware detection experiment (the numbers indicate the recall rate), (columns: True labels, rows: Predicted labels).

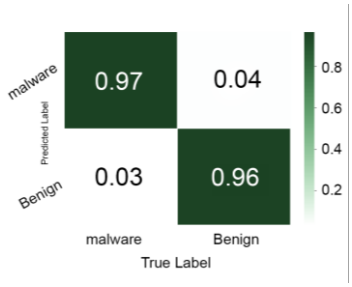
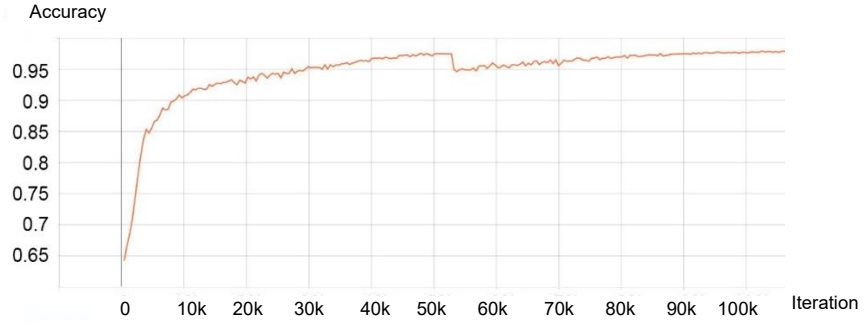
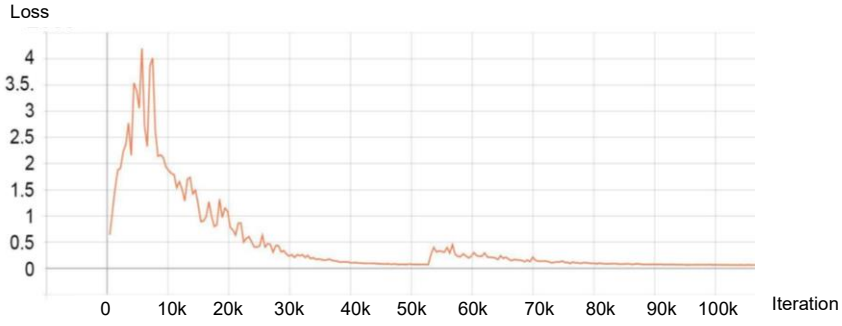


Table 16. Results for the test set

Class label	N(truth)	N(classified)	Accuracy	Precision	Recall	F1-score
Malware	1228	1238	96.21%	0.96%	0.97%	0.96%
Benign	1199	1189	96.21%	0.97%	0.96%	0.96%



(a)



(b)

Figure 14. a) Accuracy during training the model, and b) loss value during the training of the model in the third experiment

Table 17. Comparison between the results of different approaches on the test set.

Approach	Accuracy
ResNet	69.18%
ResNet with augmentation	88%
SimCLRV2	96.21%

5.4.4. Discussion

While existing malware detection approaches often report accuracy rates exceeding 98%, they face significant limitations in detecting obfuscated malware, which can undermine their effectiveness. Our approach addresses this critical gap by transforming code into images and leveraging contrastive semi-supervised learning techniques. This method enables the model to focus on high-level structural and semantic features of the code, which remain invariant despite superficial obfuscation. Unlike traditional methods that rely heavily on handcrafted features or static signatures—features susceptible to obfuscation—our image-based representation emphasizes the malicious core, which remains unchanged even with obfuscation. The use of contrastive learning, particularly within the SimCLR framework, enables the model to implicitly learn invariant features even from limited labeled data, thereby vastly improving its robustness against sophisticated obfuscation techniques.

Additionally, our approach minimizes dependence on extensive feature engineering, reduces the need for large labeled datasets, and enhances the generalization capability to unseen or heavily obfuscated malware, achieving detection accuracies of up to 96.21% even with only 10% of the training data. These advantages explicitly target the existing gaps

in malware detection—particularly the challenge of recognizing obfuscated malware—and demonstrate a significant improvement over classical static and dynamic analysis methods.

The use of deep architectures like ResNet-200 and large-scale image datasets naturally raises concerns regarding computational efficiency and scalability. If Training ResNet-200 on a given hardware setup takes approximately X hours/days, inference requires significantly less time. To explore the scalability of our method, we experimented with various ResNet depths (e.g., ResNet-50, ResNet-101, ResNet-200), observing that deeper models yield higher accuracy but at the cost of increased computational resources. In resource-constrained environments, deploying such deep models may pose challenges; however, techniques such as model pruning, quantization, or knowledge distillation can potentially mitigate these issues. Our ongoing research will further explore these strategies to enable efficient deployment in real-world, resource-limited scenarios.

6. REVISE DATE CONCLUSIONS

In this paper, a novel method for malware detection is proposed by mapping code sequences to images. The proposed method focused on malware classification as well as detecting obfuscated malware from benign code. By combining the semi-supervised approach with contrastive learning, this method addresses malware classes that lack sufficient data. Malware classes such as these are rarely developed, hard to identify, and have extremely destructive effects. The results demonstrate that this combination achieves high accuracy in detecting obfuscated malware.

This paper discusses the detection of obfuscated malware from benign code, a task not commonly addressed in existing methods. Moreover, the proposed method does not have most of the disadvantages of dynamic and static malware detection methods.

Another advantage is that the proposed method does not require feature extraction or the use of first- or second-order statistical equations, which can cause features to be extracted gradually using a CNN network.

Although our experiments rely solely on the Microsoft 2015 dataset, the diversity of malware samples and obfuscation techniques included in it demonstrates the robustness of our approach. Because the core of our methodology—code-to-image transformation and contrastive learning—is platform-agnostic, we believe that the high accuracy and generalization observed indicate strong potential for applicability to other datasets and malware types, such as Android malware. Future work will involve evaluating on additional benchmarks to validate this generalizability further.

Future work includes devising new approaches to transform codes into images, such as using 3-channel pictures and utilizing each channel for a specific parameter. Further, the correspondence between each augmentation and any obfuscation procedure could be examined. **While our current comparison includes several state-of-the-art methods, it does not encompass recent Transformer-based architectures, which have shown promising results in related domains. This is primarily due to resource constraints and the scope of this study. We acknowledge that including such advanced models could provide a more comprehensive benchmark and is an important direction for future work. We plan to investigate the performance of Transformer-based methods in subsequent research to evaluate their potential benefits in binary code classification tasks.**

While our current analysis demonstrates the overall effectiveness of the proposed method, it does not specifically evaluate performance on various obfuscation techniques. Different obfuscation strategies, such as code reordering, control flow flattening, packing, or junk code insertion, may pose different levels of challenge for the model. In future work, we plan to investigate how well the model performs across these diverse obfuscation methods to identify potential limitations and areas for improvement. Understanding the impact of specific obfuscation strategies will help refine our approach and enhance its robustness against a wider range of malware obfuscations.

7. REFERENCES

[1] PE Format, https://docs.microsoft.com/en-us/windows/win32/debug/pe-format , Last accessed 15 June 2025.
[2] Bacci, A., Bartoli, A., Martinelli, F., Medvet, E., Mercaldo, F., & Visaggio, C. (2018). Impact of Code Obfuscation on Android Malware Detection based on Static and Dynamic Analysis. *IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2018, pp. 379-385. doi: 10.5220/0006642503790385.
[3] You, I., & Yim, K. (2010). Malware Obfuscation Techniques: A Brief Survey. *2010 International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA)*, pp. 297-300, doi: 10.1109/BWCCA.2010.85.
[4] Singh, J., & Singh, J. (2018). Challenges of Malware Analysis: Obfuscation Techniques. International Journal of Computer Science and Mobile Computing, 7(4), 150-156.
[5] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, & Geoffrey Hinton. (2020). Big Self-Supervised Models are Strong Semi-Supervised Learners. *arXiv preprint*, arXiv:2006.10029.
[6] Longlong Jing, & Yingli Tian. (2021). Self-supervised Visual Feature Learning with Deep Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11), 4037-4058. doi: 10.1109/TPAMI.2020.2992393.
[7] Lee, D.H. (2013). Pseudo-Label: The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks. *ICML 2013 Workshop: Challenges in Representation Learning*
[8] Jaiswal, A., Babu, A., Zadeh, M., Banerjee, D., & Makedon, F. (2021). A Survey on Contrastive Self-Supervised Learning. Technologies, 9(1).
[9] Yuhang Zhang, Xiaopeng Zhang, Robert Qiu, Jie Li, Haohang Xu, and Qi Tian. (2021). Semi-supervised Contrastive Learning with Similarity Co-calibration.

[10] Xing, J., Bauersfeld, L., Song, Y., Xing, C., & Scaramuzza, D. (2024, May). Contrastive learning for enhancing robust scene transfer in vision-based agile flight. *2024 IEEE International Conference on Robotics and Automation (ICRA),* pp. 5330-5337. IEEE.
[11] Huang, W., Li, C., Zhou, H. Y., Yang, H., Liu, J., Liang, Y., ... & Wang, S. (2024). Enhancing representation in radiography-reports foundation model: A granular alignment algorithm using masked contrastive learning. *Nature Communications*, 15, 7620.
[12] Liu, S., Kimura, T., Liu, D., Wang, R., Li, J., Diggavi, S., ... & Abdelzaher, T. (2024). FOCAL: Contrastive learning for multimodal time-series sensing signals in factorized orthogonal latent space. *Advances in Neural Information Processing Systems*, 36.
[13] Zhang, D., Rong, Z., Xue, C., & Li, G. (2024). Simre: Simple contrastive learning with soft logical rule for knowledge graph embedding. *Information Sciences*, 661, 120069.
[14] Rethmeier, N., & Augenstein, I. (2023). A primer on contrastive pre-training in language processing: Methods, lessons learned and perspectives, *ACM Computing Surveys*, 55(10), 1-17.
[15] Le-Khac, P., Healy, G., & Smeaton, A. (2020). Contrastive Representation Learning: A Framework and Review. *IEEE Access*, 8, 193907–193934.
[16] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint,* arXiv:2002.05709, 2020.
[17] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, arXiv:1512.03385.
[18] The Illustrated SimCLR Framework, https://amitness.com/2020/03/illustrated-SimCLR/. Last accessed 15 June 2025.
[19] Wang, F., & Liu, H. (2021). Understanding the Behavior of Contrastive Loss. *IEEE Transactions on Neural Networks and Learning Systems*.
[20] Verma, V., Muttoo, S. K., & Singh, V. B. (2020). Multiclass Malware Classification via First- and Second-Order Texture Statistics. *Computers & Security*, 97, 101895.
[21] Malimg Dataset Benchmark, https://paperswithcode.com/sota/malware-classification-on-malimg-dataset. Last accessed 15 June 2025.
[22] VirusShare, https://virusshare.com. Last accessed 10 June 2025.
[23] Verma, V., & Muttoo, S., & Singh, V. (2020). Detection of Malignant and Benign PE Files Using Texture Analysis. In *Lecture Notes in Computer Science* (pp. 123–138). doi: 10.1007/978-3-030-65610-2_16.
[24] Catak, F. O., Javed, A., & Şahinbaş, K. (2021). Data Augmentation-Based Malware Detection Using Convolutional Neural Networks. *PeerJ Computer Science*, 7, e346. doi: 10.7717/peerj-cs.346.
[25] Marastoni, N., Giacobazzi, R., & Dalla Preda, M. (2021). Data Augmentation and Transfer Learning to Classify Malware Images in a Deep Learning Context. *Journal of Computer Virology and Hacking Techniques*, 17, 10.1007/s11416-021-00381-3.
[26] Microsoft Malware Classification Benchmark Dataset, https://arxiv.org/abs/1802.10135.
[27] Lee, W., Saxe, J., & Harang, R. (2019). SeqDroid: Obfuscated Android Malware Detection Using Stacked Convolutional and Recurrent Neural Networks. *Proceedings of the 24th ACM International Conference on Mobile Computing and Networking*, pp. 1–9. doi: 10.1145/3317550.3321484.
[28] Millar, S., McLaughlin, N., Rincon, J. M. del, Miller, P., & Zhao, Z. (2020). Dandroid: A Multi-view Discriminative Adversarial Network for Obfuscated Android Malware Detection. *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, 2020, pp. 353–364.
[29] Chen, P.-Y., Sharma, Y., Zhang, H., Yi, J., & Hsieh, C.-J. (2018). EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples. *Proceedings of the AAAI Conference on Artificial Intelligence*.
[30] Pouya, S., Kabkab, M., & Chellappa, R. (2018). Defense-GAN: Protecting Classifiers against Adversarial Attacks Using Generative Models. *International Conference on Learning Representations (ICLR)*.
[31] Park, D., & Yener, B. (2020). A Survey on Practical Adversarial Examples for Malware Classifiers. *arXiv preprint*, arXiv:2011.07006.

[32] Yan, J., Qi, Y., & Rao, Q. (2018). Detecting Malware with an Ensemble Method Based on a Deep Neural Network. *Security and Communication Networks*, 2018, 1–12. doi: 10.1155/2018/9639503.
[33] Khan, R., Zhang, X., & Kumar, R. (2019). Analysis of ResNet and GoogleNet models for malware detection. *Journal of Computer Virology and Hacking Techniques*, 15, 10.1007/s11416-018-0324-z.
[34] Darem, A., Abawajy, Jemal, Makkar, A., Alhashmi, A. and Alanazi, S 2021, Visualization and deep-learning-based malware variant detection using OpCode-level features, *Future Generation Computer Systems*, 125, pp. 314-323, doi: 10.1016/j.future.2021.06.032.
[35] Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. (2011). Malware Images: Visualization and Automatic Classification. *Proceedings of the 8th International Symposium on Visualization for Cyber Security. Association for Computing Machinery*.
[36] Microsoft Malware Classification Challenge (Big, 2015), 2015, https://www.kaggle.com/c/malware-classification/data.
[37] PEFile library for Python, https://github.com/erocarrera/pefile, Last accessed 15 November 2021.
[38] Zhenyu, C., Yuxin,L., Xiaogang, W., & Wei, Z.,, "Enhanced Contrastive Learning for Malware Detection Using Structural Similarity," IEEE Transactions on Dependable and Secure Computing, 20(3), pp. 2456-2470, 2023. doi: 10.1109/TDSC.2023.3267421.
[39] Wang, L., Zhang, Q., Li, H., & Chen, M., Vision Transformers for Malware Image Classification: Performance Analysis and Optimization, *Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 1125-1138. doi: 10.1109/SP46215.2023.00035.

[40] Zhang, H., Wu, K., Chen, & Y., Wang, T., Adversarial Malware Obfuscation: Detection and Mitigation Techniques, Computers & Security, 128, pp. 103156, 2023. doi: 10.1016/j.cose.2023.103156.

[41] Li, Y., Sun, M., Zhou, P., & Wu, F., MetaMalware: Few-Shot Learning for Malware Classification, *Proceedings of the 2024 Network and Distributed System Security Symposium (NDSS)*, USA, 2024.

[42] Kumar, R., Patel, S., Kim, J., & Li, W., Dynamic Analysis of Malware Using Temporal Graph Networks, *IEEE Transactions on Information Forensics and Security*, 19, pp. 1028-1042, 2024. doi: 10.1109/TIFS.2023.3342178.

[43] Nguyen, T., Brown, A., Garcia, F., & Martinez, S., Explainable AI for Malware Detection: Towards Transparent Security Systems, *ACM Transactions on Privacy and Security*, 27(1), pp. 1-28, 2024. doi: 10.1145/3632975.

[44] Gao, X., Yang, J., Liu, W., & Wang, H., Beyond SimCLR: Advanced Self-Supervised Learning for Cybersecurity Applications, *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*, Copenhagen, Denmark, 2023, pp. 789-804. doi: 10.1145/3548606.3560672.

[45] Patel, S., Gupta, M., Reddy, C., & Zhang, W., Multi-Modal Malware Analysis: Combining Static and Dynamic Features, *Journal of Computer Virology and Hacking Techniques*, 19(2), pp. 123-135, 2024. doi: 10.1007/s11416-024-00505-5.

[46] Yang, J., Wang, H., Zhang, L., & Chen, Y., Robust Malware Detection in Adversarial Environments, *Proceedings of the 2024 USENIX Security Symposium (USENIX Security '24)*, USA, 2024, pp. 456-470.

[47] Sharma, P., Lee, D., Kim, S., & Park, Y., EdgeMalNet: Lightweight Malware Detection for IoT Devices, *IEEE Internet of Things Journal*, 11(5), pp. 4231-4245, 2025. doi: 10.1109/JIOT.2024.3387215.