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

Mohammad Mahdi Maghouli, Mohamadreza Fereydooni, Monireh Abdoos and Mojtaba Vahidi-Asl

Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran
{m.maghouli, m.fereydooni}@mail.sbu.ac.ir, {m_abdoos, mo_vahidi}@sbu.ac.ir

Abstract:
With the advent of new technologies, using various formats of digital gadgets is becoming widespread. In today's world, where everyday tasks are inevitable without technology, this extensive use of computers paves the way for malicious activity. As a result, it is important to provide solutions to defend against these threats. Malware is one of the well-known and widely used means utilized for doing destructive activities by malicious attackers. Producing malware from scratch is somewhat difficult, so attackers tend to obfuscate existing malware and prepare it to become an unrecognizable program. Since creating new malware from an old one using obfuscation is a creative task, there are some drawbacks to identifying obfuscated malwares. 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, which simulate some obfuscation changes and combine them to generate complex ambiguity procedures, our proposed solution is able to construct, learn, and detect a wide range of obfuscations. This work addresses two issues: 1) malware classification despite the data deficiency and 2) obfuscated malware detection by training on non-obfuscated malwares. According to the results, the proposed method overcomes the data shortage problem in malware classification, as its accuracy is 90.1% when just 10% of data is used for training the model. Moreover, training on basic malwares 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 of information security and communication networks. Automatic malware detection is important for users to find ways to deal with the malicious effects of malwares. In recent years, machine learning methods have become promising and effective techniques, which are successfully used for malware detection. 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 for dealing with the shortage of labeled data.

In general, malware detection methods can be summarized into three categories: static analysis, dynamic analysis, and hybrid analysis. Statistical analysis tries to detect malwares using the program's code, which is converted from a binary file to assembly code. This technique analyses Portable Executable (PE) [1] files without running them and uses the code's opcodes, control flow graph, function call frequency, or system-level APIs.

It is clear that 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 uncertainty in this conversion increases the complexity of this method [2].

Dynamic analysis tries to find malicious behavior by running the program in a controlled environment such as virtual machines or sandboxes. 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 some access. However, it cannot be claimed that the environment can follow the entire behavior of the program. In spite of these limitations, dynamic analysis tries to detect malware using a sequence of API calls and system calls and classifies them into common categories such as Ransomware, Worm, 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 use the same logic, or in some cases, reusing code or using malicious libraries are observed, the code can be classified as malware or benign.

The main challenge of malware detection is related to obfuscated malware 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 the appearance of the code is. Therefore, it is less likely to be detected.

Different techniques such as disassembling and reassembling, repackaging, data encoding, and changing the code order are used for obfuscation [3, 4]. The greater the obfuscation of the code, the more difficult it is to detect malware for antivirus programs that statically scan for threats.

Another point is that obfuscated malware cannot be categorized in several specific ways since this process depends on the creativity and innovation of the attackers in such a way that the appearance of the code changes, but its logic and primary purpose remain the same. In this regard, the obfuscation of code can be compared to the noise in the images. Therefore, if we can ignore the obfuscation of the code, which does not change the identity and nature of the program, we can always identify 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 main 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 in the same way. 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 a malicious code, no matter how ambiguous the code is. Moreover, it can also detect a large number of obfuscated codes.
generated from the main malware just by using a small number of the original malware code. By achieving a higher abstraction of implemented code, we can identify what the code is, just as we ignore specific details of a landscape with a general vision. Obviously, Computer Vision methods have been maturing dramatically over the last few years with 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 is able to identify the core concept of giving samples and discards the details that do not contribute to the identity of the image. We can utilize this method to detect the malicious or non-malicious core of the software. Moreover, contrastive learning, which is introduced in SimCLR [11] as a semi-supervised method, has been used for malware classification and obfuscated malware detection in this paper.

This paper is organized as follows: Basic concepts are described in Section 2. The literature reviews and the proposed method are included in Section 3 and Section 4, respectively. The experimental results in two parts including malware detection and obfuscated malware detection are given in Section 5. Finally, the paper is concluded in Section 6.

2- Basic Concepts

In this paper, a malware classification and an obfuscated malware detection method are proposed using a semi-supervised approach combined with contrastive learning, which is applied to the images corresponding to the codes. In the following, semi-supervised learning and contrastive learning are described as basic concepts.

2-1- Semi-supervised Learning

Semi-supervised learning methods are a class of machine learning techniques in which both unlabeled and labeled data are used 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.

There are various solutions to implement and utilize semi-supervised learning methods which are discussed in [6]. One of the most prominent ways is to train a model with a small fraction of labeled data then use huge amounts of unlabeled data to generate pseudo-labels for every input using trained networks. In order to define the loss function, we can consider the loss between pseudo-label and predicted probability obtained by passing data through the network and fine-tuning it [7].

Semi-supervised learning can also be applied by first training the network with unlabeled data in a self-supervised way by performing augmentations on input data [8]. In this regard, we need a proper loss function to classify most similar but unlabeled inputs to the same group, so we use a contrastive loss function (more details in Section 2-2). The network will be fine-tuned by using a small portion of labeled inputs. 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 which 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 learning representations and features from the data, and contrastive learning could be beneficial in these cases. 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. There are several applications for contrastive learning in different domains some of which discussed in [10].

A framework for contrastive learning of visual representations, called SimCLR, has been introduced in [11]. This framework uses a semi-supervised learning approach by utilizing image augmentation for automated labeling images. When an image is converted to another one, the core concept of the image does not change, and therefore, both images should have the same label when contrastive learning has been used. Data are passed to an encoder network, and the output of each image is a feature vector. This framework specifically uses ResNet [12] as an encoder [11], as can be seen in Fig. 1, taking an image and generating a 1-hot feature vector as an output known as an “encoder unit”. The size of the feature vector is 2048 indicated by $h$ in Fig. 1. These feature vectors are then given as an input to another fully connected neural network, which uses a non-linear function [11].

![Fig. 1. The SimCLR framework [13]](image)

The learning steps are performed in 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 Fig. 2.

![Fig. 2. Contrastive learning](image)
It should be noted that this framework outperforms the state-of-the-art methods for ImageNet data set classification. Moreover, this framework can overcome the lack of enough images in some applications since it can generate new data from existing samples. This framework claims that using only 10% of ImageNet data for training, it can achieve an accuracy which is about 80% of the accuracy reached when a large portion of the labeled data is used for training [11].

SimCLR-V2 is an enhanced version of SimCLR based on contrastive learning extended by Google [5]. Network training is performed in three steps in this framework; in the first stage, an unsupervised pre-training with a task agnostic approach trains a large network regardless of the labels of data in such a way that the data differentiation is maximized. Then, it discards half of the projection head and defines a new projection head on the 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 large network obtained is used to train a smaller network, in a process called distillation. The steps of this framework are shown in Fig. 3, and the pseudo code of SimCLR framework is shown in Algorithm 1.

![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 $i \leftarrow 1$ to $\text{epoch\_size}$ do

foreach batch $b$ in AllBatches do

features =

    make\_two\_transforms\_for\_each\_data\_using\_data\_augmentation\_on\_each\_image\_in\_batch();

    hidden\_Layers\_Output = ResNet(depth, features);

    projection\_Head\_Output = model.projection\_head(hidden\_Layers\_Output);

    calculate\_Contrastive\_Loss(projection\_Head\_Output, b, datas);

    model = tuning\_projection\_Head\_using\_constrastive\_Loss();

end

fine\_tuning\_fully\_connection\_layer\_using\_supervised\_data();

end
distilled\_Model = distilling\_model\_in\_a\_task\_specific\_way(model, dataset);
Regardless of the label of the data, the pre-training of the network is to select a batch of data, apply two augmentations to each image, and create two augmented images as depicted in Fig. 4. Consequently, the amount of training data per batch is twice the size of the batch size. Each obtained image is then passed through an encoder, here ResNet101, and its feature vector is extracted.

![Fig. 4. Two augmented samples](image)

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).

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

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 are provided in [14]. 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 increase the understanding of the model from the malware core of the code and overcome the challenge of obfuscation in the code. Another challenge of detecting obfuscated malware is the large volume of obfuscated malware compared to their underlying malware. On the other hand, SimCLR, by defining a new contrastive loss function, offers a semi-supervised method such that a solution can be provided to the challenge of data deficiency and the 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 number of unlabeled data.
3- Literature Review

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

A method for malware classification introduced by Verma et al. in [15]. They convert malware code to an image and then use first-order and second-order statistical equations to extract the features using the distribution and correlation of different points on a one-channel image. Malimg [16] is a proper dataset in this regard, which is used in this work besides the dataset gathered from [17]. By manually extracting features and analyzing them statistically, codes are classified into different malware classes. The result was 98.04% for precision, 98.06% and 98.05% for recall and F1 measure, respectively. However, they did not use any obfuscated malware. Since the features have not been extracted automatically, it leads to lose a large number of discriminative features. The authors extended their work on malware detection to focus on false-negative error reduction. Some new statistical features have been used to improve the accuracy on a dataset, including 10000 data, without any obfuscated data [18].

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

Convolutional Neural Networks (CNNs) has been used for malware classification on codes that are converted to three-channel images in [19]. 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 a class of benign data, malwares are categorized into different classes. However, obfuscated malwares have not been used explicitly in their method, which is much more challenging to detect [19].

Obfuscation has been used to overcome the problem of data deficiency in [20]. The authors tried to increase the amount of data using a mapping from a generated code to an image and utilizing augmentation on the image. Then, transfer learning has been used on a pre-trained network for malware classification. They achieved 93.8% accuracy on Microsoft2015 Dataset [21] and 98.5% accuracy on Malimg Dataset.

Obfuscated malware detection has also been studied in [22] and focused 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 (RNN) and CNNs in order 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. [23] presented a method for malware detection on Android. Four common obfuscation methods are used: Presence of a selection of API calls and Android commands, permissions, and opcode instructions. Discriminative Adversarial Network (DAN) is used to train the network to classify obfuscated malwares from benigns. 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 [23].

In recent years, obfuscated malware detection has been considered in kinds of literature. This type of malware, which is derived from other malwares, forms a huge number of malwares generated every day, which is widely used for some 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 obfuscated malware detection. After training the network, the probability distribution of occurrence of any point is estimated by a generative network, and obfuscated malwares are detected using this estimated value. Due to the creative nature of obfuscation, generative networks cannot consider all the possible obfuscation, and therefore, the goal of obfuscated malware detection cannot be fully achieved.

Chen et al. [24] 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 in order to maintain against these types of attacks [25]. The vulnerability of deep neural networks against adversarial changes has been addressed in [26]; while stating the types of attacks, it has been shown that malware can bypass detection by using this approach.

In [27], an ensemble method called MalNet has been introduced for malware detection, which achieved 99.36% accuracy on the Microsoft2015 dataset [21]. 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 [28] in which byte codes are directly converted to images for training. The model achieved 88% accuracy.

Daram et al. proposed an ensemble method for malware classification [29]. Two models were ensembled for the classification of malwares on the Microsoft2015 dataset [21]: in the first one, static features such as file size and n-gram opcodes have been extracted and used for the XGBoost model; in the second one, the executable files 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.

Table 1. Summary of related works

| Authors          | Approach                        | Method            | Accuracy | Dataset               |
|------------------|---------------------------------|-------------------|----------|-----------------------|
| N. Marastoni et al. 2021 [20] | Malware classification | LSTM + CNN       | 93.8 %   | Microsoft 2015        |
|                  |                                 |                   | 98.5%    | Malimg                |
| A. Deram et al.2021 [29] | Malware classification | Ensemble         | 99.12%   | Microsoft 2015        |
| Khan et al. 2018 [28]  | Malware classification | ResNet 101       | 78.84%   | Microsoft 2015        |
|                  |                                 | ResNet 152       | 88.36%   |                       |
| Yan et al. 2018 [27]   | Malware / benign detection | MalNet           | 99.13%   | Microsoft 2015 +     |
|                       |                                 |                  |          | collected benign samples|
| F. Catak et al. 2020 [19] | Malware classification | CNN              | 96%      | Mal-API-2019           |
| V. Verma et al. 2020 [18] | Malware / benign detection | Texture statistics | 99.61%   | Custom dataset         |
| W. Lee et al. 2019 [22]  | Malware classification | CNN-RNN-PA       | 99.86%   | VirusTotal             |
| V. Verma et al. 2020 [15] | Malware classification | CNN              | 98.58%   | Malimg                |
4- The Proposed Method

As mentioned before, to overcome the challenge of detecting obfuscated malware, it is important to ignore their 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. In this regard, the code should be mapped to an image to make a higher-level abstraction of the binary codes, i.e., minor changes do not affect the general view of the image. Therefore, the code to image conversion implicitly overcomes small obfuscations. Image augmentation is usually used to perceive the identity of the images in semi-supervised methods. Augmentation also helps generate new samples from an image to address the lack of data problem. Each method of augmentation can be considered as an obfuscation approach. For example, vertical flip corresponds to the obfuscation of malware, which is created by moving parts of code or reordering in a way that does not affect the logic of the main program. Even a few degrees of blur also can be considered as changing some details without altering the general concept of the image, which could be analogous to adding or removing some useless lines of code to 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 between instances of different classes and by maximizing the similarity between instances of the same class. As a result, the proposed method is a combination of contrastive learning and semi-supervised learning in which new samples are generated using augmentation and classified as described.

The general framework of our proposed method is shown in Fig. 5. First, two sets of benignware and malware are collected. Benignware data is collected from Windows system files and cleaned so that it is similar to malware format. The binary codes of the windows operating system are selected because it contains API calls and low-level system calls as well as high-level permissions that make them difficult to detect from malwares.

Malware data is collected from the Microsoft2015 dataset [21], which is presented in nine classes as detailed in Section 5. One of them is called Obfuscator.ACY contains only obfuscated malwares, and the other classes include primary malwares in each class (malware with no obfuscation). This data is then converted to grayscale images so that every 8 bits correspond 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 for the first problem focuses on the lack of data, once training with only 20% and 10% of the data and testing on the remaining 80% and 90%, respectively, which is described in the next section.

4-1- Converting codes to images

In this method, the bytecode of the software converts to an image. The bytecode is reshaped to a binary image as described in [30], i.e. a two-dimensional array with the size $1024 \times h$ in which $h$ depends on the length of the code. Fig. 6 illustrates an executable bytecode sample and Fig. 7 shows an example of an image obtained from a bytecode as described in Algorithm 2.
4-2- Malware Classification

First, a large 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 it can be seen in the figure, ResNet101 is used as an encoder.

![Fig. 5. Illustration of the proposed method](image-url)
Fig. 6. A malware bytecode sample

Fig. 7. Result of converting a code to an image [30]

Algorithm 2: Binary Code to Image Converter

Result: 1024 × h PNG images
prepare binary code files in dataset;

\[
\text{foreach } \text{file } f \text{ in bytecodesDataset do}
\]
\[
\text{bytes} = f.\text{readBytes}();
\text{width} = 1024;
\text{height} = \text{len(bytes)}/\text{width};
\text{image} = \text{bytes.reshape(width, height)};
\text{saveBytesAsPNG(image)};
\]
\[
\text{end}
\]

According to the results reported in [5], if the problem involves a shortage of data, utilizing deeper networks along with larger batch sizes could result in a better outcome. Therefore, Resnet200 is used to train the network by 20% of data and to evaluate the remaining 80% in the first experiment and by 10% of data for training and 90% for evaluation in the second experiment, which is depicted in Fig. 8. The generated feature vectors from the encoder unit then pass through the projection head and tune the projection head nodes in 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.
4-3- Obfuscated malware detection

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

1. The train set which contains 80% of the whole benign-ware and all the basic malwares (without any obfuscation).

![Fig. 8. Illustration of the train/test splitter module in the malware classification](image)

2. The test set which contains the remaining 20% of benign wares and the whole Obfuscator.ACY class in Microsoft2015 dataset, which contains only obfuscated malware as illustrated in Fig. 9.

![Fig. 9. Illustration of the train/test splitter module in the obfuscated malware detection problem](image)

For malware data, all the data in the Microsoft2015 dataset is used for training the network, except the Obfuscator.ACY, which is used for the test phase. Therefore, the model is tested with a category of data, which is never seen before that is obfuscated malware. We demonstrate that we
can predict many obfuscated malwares from benign codes through our experiments. The obfuscation in a code could be seen as augmentation in an image, so our network learns how to ignore obfuscation in corresponding images of codes and determine whether the code is malicious or not. For generalization, using the SimCLR framework, some augmented images are generated from the base images in the training set, and 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 act as obfuscation in the corresponding code. According to Fig. 4, using the SimCLR framework, two augmented images are generated from each image in the batch in the training module. In order to accomplish this, we first resize the shorter side of the image to 224 pixels and resize the other side in such a way that the ratio of the image preserves. Then, 224×224 random crops are generated and an augmentation is randomly selected and applied. As a result, two output images with a different combination of augmentations are achieved from each input image. The desired augmentation includes random flip, color distorts, reshape and blur. Fig. 10 shows an example of augmentation.

![Augmented Images](image.png)

*Fig. 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 a 2048-dimensional vector extraction as a feature vector. The feature vectors are given to a fully connected network in the projection head to train the weight of the network. The trained network is tested on a dataset including obfuscated malware and benign wares. Data preprocessing is applied to test images by resizing them to 250 pixels on the shorter edge while maintaining the ratio and cropping them to 224×224 as part of the test phase.

### 5- Experimental Results

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

#### 5-1- Dataset

Microsoft Malware Classification Challenge dataset (Microsoft2015) [21] has been used for the evaluation of the proposed methods. This dataset includes the byte-code of malware classified in nine malware classes used for a contest held by Microsoft on Kaggle in 2015 [31]. The classes
are not balanced in terms of data distribution, which causes the classification to be more challenging. The number of samples in each class varies from 42 to 2942 samples. This dataset has been widely used in [20, 27-29] and challenges. The details of the samples in the dataset are given in Table 2.

Table 2. Malware classes in Microsoft2015 Dataset [21]

| 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 kind of obfuscated malware |
| Gatak          | 1013                       | Backdoor        |

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

Additionally, because of the lack of proper standardized benign datasets, Windows system files are used as benign codes since they call low-level system APIs just as malware does. Therefore, they are more difficult to distinguish from malware which can be used for evaluation of the proposed method. For this purpose, the data are collected first and cleaned in such a way that their structure is similar to 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 [32].

After code preparation, each byte code is mapped to the brightness of a pixel; a 1-channel grayscale image is produced with a fixed width of 1024 pixels and variable height depending on the size of the code. Using one-channel images has the advantage of being able to perform the analysis with lighter 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. $T$, $F$, $P$, $N$ stand for True, False, Positive, and Negative, respectively. In Eq. (7), $\beta$ is a positive real factor.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$  \hspace{1cm} (2)

$$TPR = \frac{TP}{TP + FN}$$  \hspace{1cm} (3)

$$FPR = \frac{FP}{FP + TN}$$  \hspace{1cm} (4)

$$Precision = \frac{TP}{TP + FP}$$  \hspace{1cm} (5)
Recall = \frac{TP}{TP + FN} \quad (6)

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

5-3- Results on Malware Detection

As mentioned before, the proposed method is being evaluated using the Microsoft2015 dataset [14] that includes nine different classes of malware. To show the performance of the proposed method, three experiments have been performed. In the first experiment, 80% of the samples of each class were used in the training set while 20% of them were used as test data. In the second experiment, 20% of the samples were used for training, and 80% of the samples were considered 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 amounts of samples used for training the network and testing are given in Table 3. The hyper-parameter values in the three experiments used in the training phase are given in Table 4.

| 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 3. Distribution of data for each class of malwares on each experiment
Table 4. The network hyper-parameter values in each experiment

| Experiment | Number of epoch | 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              |

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 value of hyper-parameters used in this experiment is reported in Table 4. The accuracy and loss values during the training phase are shown in Fig. 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.

Fig. 11. a) Accuracy during training the model and b) loss value during the training the model in the first experiment
In [29], 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 [29]. The training phase has been done on a large portion of 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, the accuracy of our approach reached the accuracy of their method using CNN in training which shows the effectiveness of our proposed method when a large portion of data is used. The obtained confusion matrix and heat map are represented in Table 5 and Table 6.

5-3-2- The Second Experiment

As described before, in order to show 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 Fig. 12. In addition, the confusion matrix and heat map are given in Table 7 and Table 8, respectively.

5-3-3- The Third Experiment

In the third experiment, 10% of the labeled data were used for training the network. The network parameters are shown in Table 4. The accuracy of the proposed method using just a small number of labeled data is 90.1%. It means that by using a small fraction of data (10%), the proposed method reaches 97.75% of the best accuracy of the method presented in [20] which uses 80% of the data for training and 92.42% of the accuracy reported in [29] on the whole data. This result shows that using contrastive learning and SimCLR achieves very high detection accuracy with a few amount of data.

Given that a relatively small number of labeled data of malwares are available, this experiment shows that the proposed method is effective for malware detection not only when a small number of data is available but also when a larger data set is available. Since the number of labeled data used for training is low, contrastive learning leads to a faster and lower-cost method (in terms of data collection) for malware detection. The obtained result is illustrated in Table 9 and Table 10. Moreover, the final results in each experiment and for each metric is depicted in Table 11 and a comparison between these three experiments is given 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 to an image first, and then the deep network is trained on the basic malware and the benign using SimCLR-V2 framework.

Two main problems in obfuscated malware detection are as follows:

1- Excessive obfuscating in the code in such a way that its functionality remains the same, but its appearance is completely different.

2- A large number of obfuscated malware compared to the basic ones.
Deep convolutional neural network (DCNN) can be effectively used to cope with 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| 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 |

Table 6. Reported heat map for the first experiment, the numbers indicate the recall rate (columns: true labels, rows: predicted labels).
Fig. 12. a) Accuracy during training the model and b) loss value during the training the model in the second experiment

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).

5-4-1 ResNet without augmentation

In this method, the dataset is divided into three main parts: training set, validation set and test set. Cross-validation has been used for the evaluation of the model. ResNet101 was used in this model on images without augmentation. For training the network, the training data were divided into 5 equal parts, and in each round 4 parts were used for training while the other part was utilized for validation. The images were 224*224 for training and the best model was selected according to the accuracy achieved by the model for validation. The test data includes the unseen benign and obfuscated malware files, which achieved an accuracy of 96.27% and 69.18% on validation and test set, respectively.
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          | 97.76        | 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          | 93.59        | 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 test | Accuracy  |
|-------------------------------|----------------------------------------|--------------------------------------|-----------|
| Transfer learning [13]        | 80%                                    | 20%                                  | 93.8%     |
| Ensemble method [23]          | -                                      | 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, a set of augmentations was applied to all images to study whether augmentation on the image can act similar to obfuscation in the code or not. The goal is for the model to a better understanding the code image by utilizing augmentation in order to learn the malicious core of the data. In general, common methods of code obfuscation can be divided into the following categories:

1- Methods that are leading to changes in control flow, such as code reordering, splitting functions into two separate functions, adding switch-case structures for selecting the next basic block in the program’s control flow (Known as Flatten), etc.

2- Methods that add/remove redundant codes, such as adding and assigning useless variables, inserting useless functions without calling them, adding useless conditions or loops which do not change the logic of the code.

3- Methods that make changes the appearance of the code while preserving the logic of the code. 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. Fig. 13 shows the different kinds 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 impact on the output of the model. As a result, it can overcome the problems caused by changes in the control flow.

- Image blurring increases the noise in the image, which causes slight changes in the image without changing its concept. Although some parts of the image may be deformed, it preserves the core and the concept of the image. This is similar to changes made by the second obfuscation category.

- Color jitter includes changing the brightness, contrast, and saturation of the image. Applying these modifications to the image causes a slight change in the image appearance, but the identity of the image remains the same. This kind of augmentation is a good alternative for the third category.
These augmentations are applied to all images in each iteration randomly, and then the model is trained using the new augmented images. The only difference between this approach and the approach elaborated in Section 5-4-1 is that the original dimensions of the images are used for training and since all augmentations apply to the image, a 224*224 random resize crop image is realized 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 codes. The results show a 20% improvement in the accuracy of obfuscated malware detection in comparison to the previous approach. According to the results, we can claim that the hypothesis presented in this paper is accurate, and image augmentation helps detect obfuscated malware and propose a proper solution to one of the main problems of malware detection. Algorithm 3 shows the pseudocode of the image augmentation methods.

### Algorithm 3: Image Augmentation on Batch

**Result:** augmented image

```plaintext
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 codes. The accuracy of the method is 98% and 96% on the training set and test set, respectively, while in the ResNet with augmentation method, which is provided in the Section 5-4-2, achieved an accuracy of 89.23% in 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 hyper-parameter values used in the experiment are given in Table 13. In addition, the obtained confusion matrix and heat map are represented in Table 14 and Table 15, respectively.

**Table 13. Hyper-parameters used in the experiments**

|                     | Number of 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%    |
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%   |

6- Conclusions

In this paper, a new method for malware detection is proposed by mapping codes to images. The proposed method focused on malware classification as well as detecting obfuscated malwares from benign codes. By combining the semi-supervised approach with contrastive learning, this method overcomes some malware classes lacking sufficient data. Malware classes such as these are rarely developed, hard to identify, and have extremely destructive effects. The results show that this combination achieves good accuracy for obfuscated malware detection.
This paper discusses detecting obfuscated malware from benign code, which is not common 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 need feature extraction or use first/second-order statistical equations, which gradually causes the features to be extracted using a CNN network.

Future works include devising new approaches to transform codes into images, such as using 3-channel pictures and using every channel for a specific parameter. Further, the correspondence between each augmentation and any obfuscation procedure could be examined.

References
[1] PE Format, https://docs.microsoft.com/en-us/windows/win32/debug/pe-format, Last accessed 15 November 2021.
[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. 379-385. 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, 2010, pp. 297-300, doi: 10.1109/BWCCA.2010.85.
[4] Singh, J., & Singh, J. (2018). Challenges of Malware Analysis : Obfuscation Techniques.
[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, 2020c.
[6] Longlong Jing, & Yingli Tian. (2019). Self-supervised Visual Feature Learning with Deep Neural Networks: A Survey. in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 43, no. 11, pp. 4037-4058, 1 Nov. 2021, 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 (WREPL).
[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] Le-Khac, P., Healy, G., & Smeaton, A. (2020). Contrastive Representation Learning: A Framework and Review. IEEE Access, 8, 193907–193934.
[11] 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.
[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385.
[13] The Illustrated SimCLR Framework, https://amitness.com/2020/03/illustrated-SimCLR/, Last accessed 15 November 2021.
[14] Feng Wang, & Huaping Liu. (2021). Understanding the Behaviour of Contrastive Loss.
[15] Vinita Verma, Sunil K. Muttoo, & V. B. Singh (2020). Multiclass malware classification via first- and second-order texture statistics. Computers & Security, 97, 101895.
[16] Malimg Dataset Benchmark, 10 Tips on Writing the Living Web, https://paperswithcode.com/sota/malware-classification-on-malimg-dataset, Last accessed 15 November 2021.
[17] https://virusshare.com., Last accessed 15 November 2021.
[18] Verma, Vinita & Muttoo, Sunil & Singh, V. (2020). Detection of Malign and Benign PE Files Using Texture Analysis. 10.1007/978-3-030-65610-2_16.

[19] Catak, Ferhat Ozgur & Ahmed, Javed & Şahinbaş, Kevser & Khand, Zahid. (2021). Data augmentation based malware detection using convolutional neural networks. PeerJ Computer Science. 7. 10.7717/peerj-cs.346.

[20] Marastoni, Niccolò & Giacobazzi, Roberto & Dalla Preda, Mila. (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.

[21] Microsoft Malware Classification Benchmark Dataset, https://arxiv.org/abs/1802.10135.

[22] Lee, William & Saxe, Joshua & Harang, Richard. (2019). SeqDroid: Obfuscated Android Malware Detection Using Stacked Convolutional and Recurrent Neural Networks. 10.1007/978-3-030-13057-2_9.

[23] Stuart Millar, Niall McLaughlin, Jesus Martinez del Rincon, Paul Miller, and Ziming Zhao. Dandroid: A multiview discriminative adversarial network for obfuscated android malware detection. In Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy, pages 353–364, 2020.

[24] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, & Cho-Jui Hsieh. (2018). EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples.

[25] Pouya Samangouei, Maya Kabkab, Rama Chellappa. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. International Conference on Learning Representations, 2018.

[26] Daniel Park and Bülent Yener, "A survey on practical adversarial examples for malware classifiers,” Nov. 2020.

[27] Jinpei Yan, Yong Qi, and Qifan Rao. Detecting malware with an ensemble method based on deep neural network. Security and Communication Networks, Feb 2018.

[28] Khan, Riaz & Zhang, Xiaosong & Kumar, Rajesh. (2019). Analysis of ResNet and GoogleNet models for malware detection. Journal of Computer Virology and Hacking Techniques. 15. 10.1007/s11416-018-0324-z.

[29] 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, vol. 125, pp. 314-323, doi: 10.1016/j.future.2021.06.032.

[30] Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. (2011). Malware Images: Visualization and Automatic Classification. In Proceedings of the 8th International Symposium on Visualization for Cyber Security. Association for Computing Machinery.

[31] Microsoft Malware Classification Challenge (Big 2015), 2015, https://www.kaggle.com/c/malware-classification/data.

[32] Pefile library for python, https://github.com/erocarrera/pefile, Last accessed 15 November 2021.