A Classification System for Visualized Malware Based on Multiple Autoencoder Models

JONGKWAN LEE AND JONGDEOG LEE, (Member, IEEE)
Department of Computer Science, Korea Military Academy, Seoul 01805, South Korea
Corresponding author: Jongkwan Lee (jklee64@kma.ac.kr)

This work was supported by Cyber Warfare Research Center of Korea Military Academy and by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2019R1G1A100303013).

ABSTRACT In this paper, we propose a classification system that uses multiple autoencoder models for identifying malware images. It is crucial to accurately classify malware before we can deploy appropriate countermeasures to prevent them from spreading. Rapid malware classification is the first step in preparing effective countermeasures. Typical approaches to this problem, which can be divided into static or dynamic methods, are not suitable for efficient malware classification because they require either fixed malware patterns or lots of time to investigate, respectively. If the malware analysts have enough time and resources, they can analyze any malware thoroughly. However, finite resources mean they always suffer from a lack of time due to the malware that needs analyzing increasing at a dramatic rate. In the real world, new malware and variants of existing malware are constantly emerging. To address this issue, many researchers have developed approaches using machine learning techniques. However, to date these systems have had difficulty responding appropriately to the rapidly changing malware environment and also suffer from data imbalance problems in the training data. The system proposed in this paper consists of multiple autoencoder models that classify malware that has been converted to an image. Each autoencoder model classifies only one type of malware and is trained using only samples from the corresponding family, this allows the system to update quickly and mitigates the data imbalance problem. We demonstrate our method’s superior performance through various experiments compared to other state-of-the-art techniques using the Malimg dataset.

INDEX TERMS Autoencoder, deep learning, malware classification, malware visualization.

I. INTRODUCTION
Malware is any malicious code or program that can be harmful to computer systems. These days, various types of malware are used in attempts to damage information systems, as such, detecting and preventing malware is essential to protect these information systems. Malware classification is a widely carried out task that can be efficiently accomplished by machine learning models.

The purpose of malware analysis is to detect the malware present in order to deploy the most appropriate countermeasures for that malware type. We typically categorize malware analysis methods into two types: static and dynamic methods. Both ways have advantages and disadvantages. A static method is a signature-based approach that uses a typical pattern as the basis for detection, such as checking how similar the file hash, Shannon entropy, N-gram, or JSON structure is to known malware [1]–[4]. The static method is fast and can detect known malware with minimal false positives. However, the static method can often not detect some variants of common malware [5]. Whenever malware variants appear, new signatures must be found and continuously updated to the database by analysts. Moreover, static methods are vulnerable to obfuscation techniques, including dead code insertion, register reassignment, instruction substitution, and code manipulation designed to hide that piece of malware’s typical pattern.

On the other hand, dynamic methods are behavior-based approach that execute suspicious code in a controlled environment and monitors for any malicious behavior [6]–[10]. The dynamic method can detect new kinds of malware, malware variants, and obfuscated known malware.
However, this approach is time-consuming suffers from a considerable false-positive rate while also being subject to platform/environment dependency [5].

These methods’ common critical drawback is they both require a lot of time and effort relative to the speed malware and their mutations are being produced. Furthermore, once effective countermeasures appear, the malware will be quickly transformed to take another form. In this sense, the analysts’ task is endless while the time and human resources available are limited. Accurate, detailed analysis of all malware through static and dynamic methods is possible if we have enough time and human resources. However, only the results achieved within the limited time available are meaningful as a countermeasure. As a result, we need a new approach to save time and man hours for malware analysis and detection.

Several types of research are being conducted to overcome the problems of these conventional malware analysis methods. One of the most attractive areas of research for malware analysis is the use of deep learning. Unlike traditional machine learning, whose performance depends on the effectiveness of the identified features, deep learning does not need hand-crafted features. In other words, it automatically extracts the distinguishable features. Several deep learning models have been proposed to detect and classify malware using autoencoders [11], CNNs [12]–[14], RNNs [15], LSTM [16], or a combination of these [17].

In this paper, we adopt an autoencoder model for classification of malware. Autoencoders are unsupervised deep learning algorithms with a unique neural network structure. An autoencoder transforms the input into an output with minimal reconstruction errors. They can be widely used in many useful applications such as in data compression, visualization of higher-dimensional data, and denoising.

The basic structure of an AE consists of two parts: the encoder and decoder. The encoder, which is represented as a function \( f \), maps the input \( x \) to the output \( y \). The encoding output represents the original data’s feature while reducing the dimensions by setting the number of neurons in the encoder layer to less than the number in the input layer. On the other hand, the decoder, described as the function \( g \), maps the encoding output \( y \) to the reconstruction \( \hat{x} \). The autoencoder is trained to make the input data \( x \) and the decoding data \( \hat{x} \) as similar as possible, while never becoming exactly the same. Note that the autoencoder is a data-specific feature extractor. It is only able to extract meaningful features from data that is similar to what it has been trained on. We take advantage of these characteristics of the autoencoder.

In this paper, we present a malware classification system that uses multiple autoencoder models. We represented malware by converted them to gray-scale images before resizing them to reduce the computational burden. We trained the autoencoder models independently with separate malware images belonging to different families of malware. The reconstruction errors from each autoencoder model for the same input image are different because they have different weights and bias parameters. The error of a specific model will be relatively small if the input image belongs to the family used to train the model. Therefore, that image can be classified into the corresponding family. We adopted the Malimg dataset to analyze the performance of the proposed system through various experiments. The results show that the proposed system is straightforward to implement while being superior to CNN-based systems with a large deep learning models. Our approach is also flexible allowing it to respond to changing trends in malware quickly.

The main contribution of this paper is to present an image-based malware classification system that uses multiple autoencoder models. The proposed system does not require any malware knowledge to use it because it classifies malware only by malware images, not by any other criteria such as binary features, functions, or the structure of the malware. It quickly provides the malware family of the unknown malicious code and is easily updated with emerging malware and variants. Our system also saves analysts’ time and effort because countermeasures against malware belonging to the same family will be similar.

The rest of this paper is organized as follows. Section II presents related works while Section III describe the proposed system in detail. Section IV evaluates the performance of the proposed system in comparison with a CNN-based system. Finally, we present our conclusions in Section V.

II. RELATED WORKS

In this section, we investigate various approaches that have been used previously for the classification of malware. The classical methods for the classification of malware based on static or dynamic analysis methods is not scalable, on top of this, their performance is undermined by the dramatic increases seen in the variants of malware.

To mitigate these issues, new strategies have emerged that apply machine learning. One is the use of independent feature extraction procedures before applying machine learning algorithms. These features are then used as input data to the algorithm. The classification performance with this approach depends heavily on the extracted features. Moreover, extracting high-quality features requires expertise in malware, while the quality of features can depend to a large degree of the specific dataset tested. Another strategy involves the algorithm including a feature extraction procedure, where the original data is used as input data. This approach is independent and learns the discriminative representations from the original data.

For the first time, Nataraj et al. proposed a novel approach for visualizing and classifying malware using image processing techniques [18]. They visualized malware as a gray-scale image based on the observation that images of the same class were very similar in layout and texture. The classification task used either GIST [19], [20], based on wavelet decomposition of an image as a feature, or k-NN (k-Nearest Neighbors), a traditional machine learning algorithm. After the paper was published, many researchers became interested in this
new concept for malware classification. They also tried to apply deep learning based algorithms rather than traditional machine learning. CNN-based algorithm have proven the most popular amongst the numerous deep learning models because of the impressive image classification performance they showed.

Zhao et al. [14] proposed a malware detection method based on an improved Faster RCNN that combines transfer learning. They applied a CNN to malware images in order to acquire the representative textures. An RPN (Region Proposal Network) was also used to generate the target image frame. Since the Faster RCNN requires a large dataset, they pre-trained the Faster RCNN model with ImageNet, a database including 14 million pictures, before repurposing the trained model for malware classification to improve the performance. The method achieved 92.8% accuracy and a 6.8% false-positive rate in an experiment with 1,821 malware samples in 5 classes that were provided by the Microsoft Corporation, the samples were extended by image transformation operations such as flipping, cropping, and changing the brightness.

Yue proposes a simple and effective weighted softmax loss approach for the final layer of the deep CNN to mitigate any performance degradation caused by the unbalanced data set [21]. The original softmax loss method is not suitable for unbalanced data sets because it treats misclassification equally, which degrades model performance. The value of the weighted softmax loss for an arbitrary class reflects the relative size of that arbitrary class to the largest class. By the way, an empirically selected parameter, that takes into account the dataset’s characteristics, is used to calculate the weighted softmax values. As such, this value should be tailored for each specific dataset. Therefore, this method is not practical in the real world.

Vasan et al. [12] proposed an ensemble convolutional neural network-based architecture to classify visualized malware. They noted the fact that different CNN models provide different semantic representations of the image. VGG16 and ResNet-50 pre-trained on ImageNet were fine-tuned for malware images by transfer learning. The features extracted by these fine-tuned models were processed by classifiers such as softmax and multiclass SVM. The class of new malware images was obtained by ensembling the results from the classifiers. The experimental results with the Malimg dataset show that these kinds of ensemble methods can achieve high detection accuracy with low false alarm rates.

On the other hand, while CNN models are most commonly used for feature extraction from malware images, some studies have used other models. Kebede et al. [11] proposed a malware classification system comprised of multiple encoders that are part of an autoencoder. After training the autoencoder with the training images that were converted from malware samples, the encoder part of the autoencoder is only used in the classification system to extract features. The system then passes the extracted features through a softmax layer to determine the malware class. This system achieved 99.15% accuracy with the dataset provided by the MS Malware Classification Challenge (BIG 2015).

All the research mentioned above is basically composed of two stages. The first stage is to extract features using a deep learning model. The second stage is to classify each sample using the extracted features. The types of malware found in the real world are not fixed and vary over time. With this emergence of new types of malware, it is vital the classifications system are updated in a timely manner before the malware can spread and damages systems. Due to the data imbalances issue, the deep learning model requires enough samples belonging to the new classes to recognize the new malware class in their classification system. However, there may not be enough samples of these new malware, especially in the early days surrounding a particular malware’s emergence. We must consider this issue when designing a malware classification system that can be used successfully in the real world.

### III. PROPOSED SYSTEM

#### A. SYSTEM STRUCTURE

Fig. 1 shows the architecture of the proposed system. The input $x$ enters each autoencoder models. For each model, the reconstruction error is calculated, and the index of the model representing the minimum error is a predicted class for input $x$.

We formulate a discrete function $f_i[i]$ with values of $re_i$ for the input image $x$ as follows:

$$f_i[i] = re_i, \quad i = 0, 1, 2, \ldots, n$$  \hspace{1cm} (1)
Then, we calculate the predicted class for input $x$ as follows:

$$\hat{C}(x) = \arg \min_{i \in \mathbb{N}} f_x[i] \quad (2)$$

The argument that takes the function to its minimum is the identity of the predicted class for $x$. Table 1 shows the symbols used in the paper and their meaning.

### B. VISUALIZATION OF MALWARE

Fig. 2 shows the procedures of converting malware into an image. A given malware binary is converted to a grayscale image by reading every 8 bits as vectors of unsigned integers and converting those vectors into a 2-dimensional matrix with a fixed number of columns. The values of the matrix elements, i.e., the pixels in the image, form integers from 0 (black) to 255 (white). The size of the image depends on the size of the malware binary. The size of the malware images, when used as the input of a deep learning model, should all be the same. Interpolation techniques such as nearest-neighbor interpolation, bilinear interpolation, and bicubic interpolation are used to make different-sized images the same size. Nearest-neighbor interpolation degrades the performance of image classifiers relative to using bilinear or bicubic interpolation. Bilinear and bicubic interpolation achieve similar classifier performance. However, bicubic interpolation is much more computationally complex than bilinear interpolation. Therefore, we use bilinear interpolation to resize the malware images, due to its decreased computational complexity and strong classifier performance [22].

### C. AUTOENCODER

The AE consists of two components: the encoder and the decoder. The encoder is responsible for compressing the input $x$ into a low-dimensional representation that retains relevant information from the input, and the decoder attempts to reconstruct the input $x$ from the features extracted by the encoder [23], [24].

AEs can be created using various network structures such as feed-forward neural networks or convolutional neural networks. Furthermore, various AEs can be applied to identify each different class. In other words, our system consists of multiple types of AE that are optimized to make samples belonging to a specific family. As such, each AE is only trained with samples belonging to the one family. That is, the $i$th AE model is only trained with data belonging to the $i$th family, $T_i$. Therefore, if the input of the $i$th AE model belongs to the $i$th family, the output of the model will be similar to the input. However, if the input comes from a different family, the reconstruction error between the input and output of the model will be significant. For instance, a sample belonging to family A will have only a small reconstruction error when processed by an AE trained with samples belonging to family A. On the other hand, a sample belonging to class B will have a significant reconstruction error if processed by that same AE. We utilize these characteristic of AEs to solve classification problems.

The proposed system in this paper applies AEs based on feed-forward neural networks, but other types of neural network for AEs could also be used. The AEs for each family have different configurations and hyper-parameters to minimize their reconstruction errors for inputs from that family. Four hyperparameters need to be set before training an AE: feature size, number of layers, number of neurons per layers, and loss function.

1) **FEATURE SIZE**

This is the number of dimensions of the encoding output $y$, which is equal to the number of neurons in the middle layer. The smaller the size, the higher the compression rate for the
TABLE 2. Sample distribution in MALIMG dataset.

| Name      | class ID | # of samples | Avg. size (KB) |
|-----------|----------|--------------|---------------|
| Adialer.C | 0        | 123          | 133.10        |
| Agent.FYI | 1        | 116          | 9.50          |
| Allaple.A | 2        | 2949         | 67.25         |
| Allaple.L | 3        | 1591         | 53.41         |
| Allueron.genlJ | 4 | 198          | 85.16         |
| Autorun.K | 5        | 106          | 333.98        |
| C2Lop.P   | 6        | 146          | 314.66        |
| C2Lop.genlJ | 7 | 200          | 451.51        |
| Dialplatform.B | 8 | 177          | 10.94         |
| Dontovo.A | 9        | 162          | 14.33         |
| Fakerean  | 10       | 381          | 99.62         |
| Instantaccess | 11 | 431          | 163.29        |
| Lolyda.AA1 | 12       | 213          | 15.90         |
| Lolyda.AA2 | 13       | 184          | 15.72         |
| Lolyda.AA3 | 14       | 123          | 18.53         |
| Lolyda.AT  | 15       | 159          | 21.93         |
| Malex.genlJ | 16 | 136          | 70.49         |
| Obfuscator.AD | 17 | 142          | 149.40        |
| Rbot.gen   | 18       | 158          | 142.71        |
| Skintrim.N | 19       | 80           | 167.15        |
| Swizzor.genlE | 20 | 128          | 266.17        |
| Swizzor.genlI | 21 | 132          | 241.45        |
| VB.AT      | 22       | 408          | 304.11        |
| Wintrim.BX | 23       | 97           | 390.30        |
| Yuner.A    | 24       | 800          | 334.00        |

FIGURE 3. Basic structure of AE used in the proposed system.

We can add as many layers as we like to the AE. However, as the number of layers increases, the number of parameters to be trained increases rapidly. We select the number of layers for each AE by considering the number of available training samples and the model’s performance.

We compile the AE model using cross-entropy loss. Each pixel value in a converted malware image is in the range [0, 255]. However, we normalize this range to [0, 1] before entering the image into the input layer.

Note that feature size, the number of layers, the number of neurons per layer, and the loss functions for appropriate training depend on the data being used in the specific AE. This is a great advantage because samples belonging to different families have different image characteristics. This allows the proposed system to be updated easily.

Meanwhile, AEs are trained using only samples belonging to the target family. That is, the i-th AE model is only trained using samples belonging to the i-th family. Therefore, if the i-th AE model’s input belongs to the i-th family, the final output will be similar to the input, i.e., the reconstruction error will be tiny. However, if the input does not belong to the i-th family, the reconstruction error between the input and output is significant. This is a key idea behind the proposed classification system. Fig. 3 shows the basic structure of an autoencoder used in the proposed system.
poor performance from models using these kinds of datasets, especially for the minority classes. This issue becomes critical if the minority class is more important than the majority classes [25]. In terms of malware classification, not only majority classes but minor classes are also important. The number of samples in the minor classes tend to be small because this type of malware has not yet been widely used making it more attractive to malicious users in the future. As such, we must take appropriate countermeasures against these threats before significant damage can be done.

To evaluate our method we first split the dataset into two parts: 70% training samples and 30% testing samples in each class.

B. EVALUATION METRICS
We use accuracy, precision, recall, F1 score as evaluation metrics for the classification performance of the deep learning models tested.

1) ACCURACY
The accuracy is simply the ratio of the correctly predicted samples to the total number of samples. It is a very intuitive measure and is appropriate only when the numbers of false-positives and false negatives are roughly the same. Imbalanced datasets can easily lead to distortion of model performance. The dataset we are dealing with is imbalanced, as shown in Table 2. Thus, we must look for other metrics to evaluate the performance of the models tested.

2) PRECISION
The precision is the ratio of the correctly predicted positive samples to the total number of predicted positive samples. High precision means a low false-positive rate. This is a good measure to use when there is a high cost associated with False Positives.

3) RECALL
Recall is the ratio of correctly predicted positive samples to all samples in the actual class. It is useful to select the best model when there is a high cost associated with False Negatives.

4) F1 SCORE
The F1 score is a weighted average of precision and recall that seeks to find a balance between them. It takes both false positive and false negatives into account to evaluate the performance of the models. The classification of malware into the correct class has a high cost associated with false negatives. Note that the task under consideration here is not to determine whether some code is malicious or not, it is, instead, to determine the type of code that has already been identified or suspected as malicious in order to save malware analysts time and effort in creating anti-malware defenses. Countermeasures against malware in the same class will be similar. As such, costs are associated with both false negatives and false positives.

| Img. size | Acc. | Precision | Recall | F1 score |
|-----------|------|-----------|--------|----------|
| 32 x 32   | 0.79 | 0.78      | 0.79   | 0.78     |
| 64 x 64   | 0.94 | 0.95      | 0.94   | 0.93     |
| 96 x 96   | 0.91 | 0.91      | 0.91   | 0.90     |

False negatives and false positives lead malware analysts to take the wrong approach when trying to defend against these malicious programs. Therefore, the F1 score is a useful metric to measure the performance of a malware classification model.

As we mentioned above, the dataset we are using is highly imbalanced. We take a weighted average value of each class as the performance metric of the proposed system. We define the weighted average metrics as system metrics, including system recall and system F1 score.

Meanwhile, Top-N metrics take the N predictions with a higher probability. If one of them matches the true label, it classifies the prediction as correct. We calculate the TOP-1, TOP-2 and TOP-3 metrics for accuracy, precision, recall, and F1 score. The model then provides N prediction results to malware analysts. The analyst will then attempt to create appropriate anti-malware defense based on the provided results. If the highest predicted result is not correct, the next result is used.

C. EXPERIMENTAL RESULTS
1) IMPACT OF IMAGE SIZE
The number of neurons in the input and output layers depends on the image size, which affects the overall structure of the AE. We set the number of neurons in the hidden layers based on multiple experiments to determine the optimal number of neurons. However, we set the same latent space dimensions in order to compare the performance impact of image size. In addition, the AEs were all trained in the same way.

We resized the images to a certain fixed size to feed as inputs to the AEs. The image will not be able to contain critical information about the malware if the image size is set too small. On the other hand, the computational time and burden are only increased with little to no performance improvement if the image size is set too large.

Table 3 shows the performance according to the image size. The values presented are a weighted average of the results for each class. It can be seen that the image size is a significant impact on the performance. In the case of using a 64 × 64 image size, the performance was better than in other cases. This is only because the training dataset is suitable for the model structure used in the experiment. If different types of dataset were used, the performance results would also be different. Therefore, it would not be reasonable to use the same model structure and make the model learn in the same way. For example, when the image size and the number of data samples are small, a lightweight AE, which has small latent
space dimensions and fewer neurons to reduce parameters for optimization, would be suitable.

2) IMPACT OF NUMBER OF CLASSES
Table 4 shows the performance of scenarios in which the number of classes to be classified is different. We compared the performance of the proposed system in cases where the number of classes to be classified was 5, 10, 15, 20, and 25, respectively. As can be expected, the performance tends to degrade but this is not significant. The number of classes does not significantly affect the entire system’s performance if each AE is well trained. It is natural that the more classes there are to classify, the poorer performance the classifier will have. However, if the number of classes is maintained at a certain level, a certain level of performance can be guaranteed. Meanwhile, if images from two classes are very similar, they can be merged into the same class through analysis by malware experts. In this way, we can control any unnecessary increases in the number of classes.

3) PERFORMANCE OF SYSTEM WITH THE SAME AE
Here we investigate the performance of the proposed system consisting of AEs that have the same network structure and have been trained in the same way, regardless of samples used for training each AE. As can be seen in Table 5, the proposed system achieved more than 94% accuracy for 25 malware families, which is not a bad result. However, it is hard to say that each AE is optimized to the corresponding samples in terms of image reconstruction.

Generally, the structure of an AE depends on the complexity of features in the samples and the size of the dataset used for training. If the features are simple, it does not matter if the latent space dimensions of the AE are small. However, if not, the latent space dimensions must be large enough to represent the features present adequately. Meanwhile, if the number of samples in the dataset is large, training a complex model with many parameters to optimize classification is possible. On the contrary, it is challenging to train a model if there are not enough samples. In such a case, a simple model can be more beneficial than a complex model. As a result, each AE should be configured and trained differently according to its corresponding dataset. Based on these facts, we can improve the current system’s performance by applying AEs that are adjusted to their corresponding samples.

---

**TABLE 4.** Impact of the number of classes to classify on the performance.

| Metric | Precision | Recall | F1 score |
|--------|-----------|--------|----------|
| Class ID | 0 | 1 | 2 | 3 | 4 | 0 | 1 | 2 | 3 | 4 | 0 | 1 | 2 | 3 | 4 |
| Number of AEs | 5 | 1.00 | 1.00 | 0.97 | 0.96 | 1.00 | 1.00 | 0.99 | 0.95 | 0.86 | 1.00 | 1.00 | 0.98 | 0.96 | 0.93 |
| | 10 | 1.00 | 1.00 | 0.97 | 0.98 | 1.00 | 1.00 | 0.99 | 0.95 | 0.86 | 1.00 | 1.00 | 0.98 | 0.96 | 0.93 |
| | 15 | 1.00 | 1.00 | 0.96 | 0.98 | 0.92 | 1.00 | 1.00 | 0.99 | 0.95 | 0.86 | 1.00 | 1.00 | 0.97 | 0.96 | 0.89 |
| | 20 | 1.00 | 1.00 | 0.93 | 0.98 | 0.92 | 1.00 | 1.00 | 0.99 | 0.95 | 0.86 | 1.00 | 1.00 | 0.96 | 0.96 | 0.89 |
| | 25 | 1.00 | 1.00 | 0.92 | 0.98 | 0.92 | 1.00 | 1.00 | 0.99 | 0.95 | 0.86 | 1.00 | 1.00 | 0.96 | 0.96 | 0.89 |

**TABLE 5.** Performance of the proposed system with the same AE.

| Class ID | Precision | Recall | F1-score |
|---------|-----------|--------|----------|
| 0       | 1.00      | 1.00   | 1.00     |
| 1       | 1.00      | 1.00   | 1.00     |
| 2       | 0.94      | 0.99   | 0.97     |
| 3       | 0.95      | 0.97   | 0.96     |
| 4       | 0.95      | 0.91   | 0.93     |
| 5       | 1.00      | 1.00   | 1.00     |
| 6       | 1.00      | 0.02   | 0.04     |
| 7       | 0.66      | 0.42   | 0.51     |
| 8       | 1.00      | 1.00   | 1.00     |
| 9       | 1.00      | 1.00   | 1.00     |
| 10      | 1.00      | 0.97   | 0.98     |
| 11      | 1.00      | 0.99   | 1.00     |
| 12      | 0.98      | 0.89   | 0.93     |
| 13      | 1.00      | 0.97   | 0.98     |
| 14      | 0.97      | 1.00   | 0.99     |
| 15      | 0.97      | 0.97   | 0.97     |
| 16      | 0.58      | 0.68   | 0.63     |
| 17      | 1.00      | 1.00   | 1.00     |
| 18      | 0.94      | 0.96   | 0.95     |
| 19      | 1.00      | 1.00   | 1.00     |
| 20      | 0.38      | 0.50   | 0.43     |
| 21      | 0.40      | 0.55   | 0.46     |
| 22      | 1.00      | 0.96   | 0.98     |
| 23      | 0.86      | 0.91   | 0.89     |
| 24      | 1.00      | 1.00   | 1.00     |

Sys. precision: 0.95. Sys. recall: 0.94. Sys. F1 score: 0.93
Top-1 Accuracy: 0.9403
Top-2 Accuracy: 0.9596
Top-3 Accuracy: 0.9646
4) PERFORMANCE OF SYSTEM WITH VARIOUS AEs

As we explained in the previous section, it could be efficient to apply different AEs that have been optimized in terms of adjusting the number of neurons, image size, latent space dimensions, etc. according to the given dataset. In this section, we investigate the performance of the system, including the various AEs used to classify malware. Table 6 shows a confusion matrix that gives the detailed performance of the proposed system.

As can be seen, the proposed system achieved more than 97% accuracy for the 25 malware families, higher than that achieved by Gibert et al. who used a large CNN-based deep learning model [26]. The performance of our system has shown significant improvements compared to systems using the same AEs under the same conditions. This is because that the modified AEs in our system better represent samples in the corresponding class.

There were two sources of poor performance. One is where some malware in class 6 and 7 are misclassified as being in class 21. The both of class 6 and 7 are variants of C2Lop. Therefore, there is a tendency for these to be misclassified as they are so similar to each other. On the other hand, some malware in class 20 was incorrectly classified as being in class 21. Class 20 and 21 are also variants of the same malware, known as Swizzoer.gen. This shows our system may not be able to adequately discriminate between subclasses from the same parent class. This is because malware belonging to the same class is converted to images that appear very similar.

Nevertheless, if the malware size is small and the number of samples is large, the proposed system classifies these cases well, even if they are variations of the same malware. For example, classes 2 and 3 are both variations of Allaple. However, the F1-score for these is more than 0.99. The same goes for classes 12, 13, 14, and 15, which are small in size compared to other classes while being variants of the same malware. As a result, if the size of variants belonging to different classes derived from the same malware are large but their quantities are insufficient, combining two classes into one class can improve the classification performance of the proposed system.

Based on these observations, we experimented by combining class 6 and 7 into one class, and classes 20 and 21 into one class. After this, the accuracy is increased from 0.9775 to 0.9821. This is because combining similar classes into one class increases the number of samples and reduces the number of the target classes, which leads to an improvement in classification performance.

D. STRENGTH OF THE PROPOSED SYSTEM

The purpose of malware classification is to identify malware quickly and accurately so appropriate countermeasures can
be deployed in a timely manner. It takes time to classify malware families, unfortunately, malware analyst suffer from a lack of time due to the continuous emergence of new malware and variants. Therefore, we should reduce malware analysis burden as much as possible so that malware experts can focus on preparing countermeasures against the various types of malware. Countermeasures against malware in the same family would be similar. Therefore, fast and exact classification of malware can reduce the overall time needed to prepare effective countermeasures.

It is of note that classifying malware images is somewhat different from classifying images that do not vary significantly over time. For example, a system that classifies images of dogs and cats does not require much modification after being properly trained initially. This is because new types of dogs or cats hardly ever appear. However, new types of malware appear continuously while existing malware is constantly mutating. Therefore, systems that classify malware need to be updated continuously and quickly.

Considering these requirements, the system proposed in this paper has some distinct advantages over the typical static or dynamic analysis methods seen in conventional systems based on complex machine learning models.

First, the proposed system provides malware analysts with the malware classification results in real-time, just like any other image-based malware classification system. This approach does not require additional manual work or knowledge about malware, unlike those traditional static or dynamic analyses.

Second, the class imbalance problem can be mitigated. The AE models in the proposed system are trained independently using only samples belonging to a specific family. Therefore, there is no class imbalance problem. Classification systems that consist of one large deep learning model find it difficult to resolve this imbalance issue because the model must be trained on the available real-world datasets which are imbalanced.

Third, the proposed system has excellent flexibility. Malware families are created often and also disappear quickly over time. As such, the classification systems should be able to adapt to the changing trends seen in malware quickly. A large deep learning model must be retrained from scratch to add a new family to classify. On the other hand, the current AE models in the proposed system are unchanged when a new family is added. We simply create an AE model for the new family and insert it into the existing system without affecting other parts of the system. If a current malware family needs to be deleted, we simply remove the AE model for the corresponding class from the system while keeping the other AEs models as they are. This feature of the proposed system enables rapid responses to the ever evolving malware ecosystem.

Fourth, the proposed system achieved the best performance on the Malimg dataset compared to other state-of-the-art image-based classification approaches.

V. CONCLUSION

In this paper, we proposed a malware classification system using multiple AEs. New malware appears continuously at a great rate while known malware mutates to avoid detection systems. This means in the task of identifying malware, the classes to classify are constantly changing, unlike in other applications. Therefore, the classification system used for this purpose needs to be updated rapidly according to the changing operating environment.

Since the system proposed in this paper uses lightweight AEs to identify each malware family, only the corresponding AE needs to be modified when classes need to be added or changed. Therefore, it is possible to quickly update the system and mitigate any class imbalance problems.

To verify the practicality of the proposed system, we analyzed the performance of our system according to image size, the number of classes, etc., through various experiments based on the Malimg dataset. The proposed system achieved better classification performance than all previous methods in the literature where malware was represented as gray-scale images. We expect that the proposed system can help save time and effort, decreasing the burden on malware analysts allowing them to dedicate more resources to preparing effective countermeasures for malware.

REFERENCES

[1] D. Bilar, “Opcodes as predictor for malware,” Int. J. Electron. Secur. Digit. Forensics, vol. 1, no. 2, pp. 156–168, 2007.
[2] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, “Detecting unknown malicious code by applying classification techniques on OpCode patterns,” Secur. Informat., vol. 1, no. 1, pp. 1–22, Dec. 2012.
[3] R. Lyda and J. Hamrock, “Using entropy analysis to find encrypted and packed malware,” IEEE Secur. Privacy, vol. 5, no. 2, pp. 40–45, 2007.
[4] C. Ravi and R. Manoharan, “Malware detection using Windows api sequence and machine learning,” Int. J. Comput. Appl., vol. 43, no. 17, pp. 12–16, 2012.
[5] S. Ranveer and S. Hiray, “Comparative analysis of feature extraction methods of malware detection,” Int. J. Comput. Appl., vol. 120, no. 5, pp. 1–7, Jun. 2015.
[6] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, “Graph-based malware detection using dynamic analysis,” J. Comput. Virol., vol. 7, no. 4, pp. 247–258, 2011.
[7] M. Ghiassi, A. Sami, and Z. Salehi, “Dynamic VSA: A framework for malware detection based on register contents,” Eng. Appl. Artif. Intell., vol. 44, pp. 111–122, Sep. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197615001190
[8] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, “A survey on automated dynamic malware-analysis techniques and tools,” ACM Comput. Surv., vol. 44, no. 2, pp. 1–42, Feb. 2012. doi: 10.1145/2089125.2089126.
[9] Z. Salehi, A. Sami, and M. Ghiassi, “MAAR: Robust features to detect malicious activity based on API calls, their arguments and return values,” Eng. Appl. Artif. Intell., vol. 59, pp. 93–102, Mar. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197616302512
[10] C. Storlie, B. Anderson, S. V. Wiel, D. Quist, C. Hash, and N. Brown, “Stochastic identification of malware with dynamic traces,” Ann. Appl. Stat., vol. 8, no. 1, Mar. 2014. doi: 10.1214/13-AOAS703.
[11] T. M. Kebede, O. Djanye-Boundjou, B. N. Narayanan, A. Ralescu, and D. Kapp, “Classification of malware programs using autoencoders based deep learning architecture and its application to the Microsoft malware classification challenge (BIG 2015) dataset,” in Proc. IEEE Nat. Aerosp. Electron. Conf. (NAECON), Jun. 2017, pp. 70–75.
[12] D. Vasan, M. Alazab, S. Wassan, B. Safaei, and Q. Zheng, “Image-based malware classification using ensemble of CNN architectures (IMCEC),” Comput. Secur., vol. 92, May 2020, Art. no. 101748.
[13] S. Yue, “Imbalanced malware images classification: A CNN based approach,” 2017, arXiv:1708.08042. [Online]. Available: https://arxiv.org/abs/1708.08042

[14] Y. Zhao, W. Cui, S. Geng, B. Bo, Y. Feng, and W. Zhang, “A malware detection method of code texture visualization based on an improved faster RCNN combining transfer learning,” IEEE Access, vol. 8, pp. 166630–166641, 2020.

[15] S. Jha, D. Prashar, H. V. Long, and D. Taniar, “Recurrent neural network for detecting malware,” Comput. Secur., vol. 99, Dec. 2020, Art. no. 102037.

[16] R. Vinayakumar, K. P. Soman, P. Poonachandran, and S. S. Kumar, “Detecting Android malware using long short-term memory (LSTM),” J. Intell. Fuzzy Syst., vol. 34, no. 3, pp. 1277–1288, Mar. 2018.

[17] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, “Malware detection with deep neural network using process behavior,” in Proc. IEEE 40th Annu. Comput. Softw. Appl. Conf. (COMPSAC), vol. 2, Jun. 2016, pp. 577–582.

[18] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, “Malware images: Visualization and automatic classification,” in Proc. 8th Int. Symp. Visualizat. Cyber Secur. (VisSec), 2011, pp. 1–7, doi: 10.1145/2016904.2016908.

[19] A. Oliva and A. Torralba, “Modeling the shape of the scene: A holistic representation of the spatial envelope,” Int. J. Comput. Vis., vol. 42, no. 3, pp. 145–175, 2001.

[20] A. Torralba, K. P. Murphy, W. T. Freeman, and M. A. Rubin, “Context-based vision system for place and object recognition,” in Proc. 9th IEEE Int. Conf. Comput. Vis., vol. 1, Oct. 2003, pp. 273–280.

[21] X. Li and W. Wang, “Learning discriminative features via weights-biased softmax loss,” Pattern Recognit., vol. 107, Nov. 2020, Art. no. 107405. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320320302089

[22] M. Song and J. Lee, “Analysis of the impact of the visualization methods on deep learning-based malware classification performance,” Korean J. Military Arts Sci., vol. 77, no. 1, pp. 511–530, 2021.

[23] D. Bank, N. Koenigstein, and R. Giryes, “Autoencoders,” 2020, arXiv:2003.05991. [Online]. Available: https://arxiv.org/abs/2003.05991

[24] J. Zhai, S. Zhang, J. Chen, and Q. He, “Autoencoder and its various variants,” in Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC), Oct. 2018, pp. 415–419.

[25] H. Kaur, H. S. Pannu, and A. K. Malhi, “A systematic review on imbalanced data challenges in machine learning: Applications and solutions,” ACM Comput. Surv., vol. 52, no. 4, pp. 1–36, Sep. 2019, doi: 10.1145/3343440.

[26] D. Gibert, C. Mateu, J. Planes, and R. Vicens, “Using convolutional neural networks for classification of malware represented as images,” J. Comput. Virol. Hacking Techn., vol. 15, no. 1, pp. 15–28, Mar. 2019.

JONGKDEOG LEE (Member, IEEE) was born in Busan, South Korea, in 1983. He received the B.S. degree in computer science from Korea Military Academy, in 2005, the M.S. degree in computer science from the University of Virginia, Charlottesville, in 2009, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign, in 2019. After his master’s degree, he was teaching basic computer science techniques to cadets at Korea Military Academy as an Instructor for five years. He has been serving Korean Army, since 2005, and is currently an Active Army officer, a Major of the Republic of Korea (ROK) Army. He has various work experience. He has served for 5th Infantry Division as a Platoon Leader and a Tactical Staff Officer. He also worked at the Army Research Laboratory (ARL) for a couple of months as an internship during his Ph.D. He is currently an Assistant Professor with the Department of Computer Science, Korea Military Academy. He has authored one book chapter and 11 international articles. His research interests include wireless sensor networks, information centric networks, the Internet of Things (IoT), and their applications.

JONGKWAN LEE was born in Seoul, South Korea, in 1976. He received the B.S. degree in electronic engineering from Korea Military Academy, Seoul, in 2000, the M.S. degree in electronic engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2004, and the Ph.D. degree in network centric warfare (NCW) engineering with Ajou University, Suwon, South Korea, in 2014. He has served as a Military Officer, since 2000, and is currently an Active Army officer, a Lieutenant Colonel of the Republic of Korea (ROK) Army. From 2011 to 2014, he had participated at the Tactical Network Research Center, Ajou University. He is currently an Associate Professor with the Department of Computer Science, Korea Military Academy. His research interests include tactical wireless networks, cyber warfare, and artificial intelligence and their applications.

* * *