IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 12, 2022, accepted January 24, 2022, date of publication February 28, 2022, date of current version March 10, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3155695

A Malware Detection Approach Using Autoencoder in Deep Learning

XIAOFEI XING[1,2], XIANG JIN[1], HAROON ELAHI[1], HAI JIANG[3], AND GUOJUN WANG[1]
[1]School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou, Guangdong 510006, China
[2]School of Information Engineering, Zhengzhou University of Industrial Technology, Zhengzhou, Henan 451100, China
[3]Department of Computer Sciences, Arkansas State University, Jonesboro, AR 72467, USA

Corresponding author: Xiang Jin (xingxf@gzhu.edu.cn)

This work was supported by the Chinese Scholarship Council under Grant No.202008440213, and the National Key Research and Development Program of China under Grant No.2020YFB1005804.

ABSTRACT Today, in the field of malware detection, the expanding limitations of traditional detection methods and the increasing accuracy of detection methods designed on the basis of artificial intelligence algorithms are driving research findings in this area in favour of the latter. Therefore, we propose a novel malware detection model in this paper. This model combines a grey-scale image representation of malware with an autoencoder network in a deep learning model, analyses the feasibility of the grey-scale image approach of malware based on the reconstruction error of the autoencoder, and uses the dimensionality reduction features of the autoencoder to achieve the classification of malware from benign software. The proposed detection model achieved an accuracy of 96% and a stable F-score of about 96% by using the Android-side dataset we collected, which outperformed some traditional machine learning detection algorithms.

INDEX TERMS Malware detection, autoencoders, malware images, mobile application security.

I. INTRODUCTION

In recent years, the rapid development of mobile internet technology has rendered the growth of the software industry. The number of malware is growing with each passing day. According to the latest China Internet Annual Network Security Report [1], as of 2019, there were as many as 13,510,900 cases of mobile Internet malware programs, with nearly 2,791,300 new cases added this year alone. The Android system has been the key to many mobile-based malware attacks due to the open nature of the Android application market. With the increase in Android malware security threats, it is necessary to develop an efficient and novel mobile malware detection method to solve the problem.

Traditional malware detection techniques are limited by the number of detection rules that need to be set manually. It is impossible to detect many new malware variants in today's world of increasing malware [2]. In recent years, malware detection techniques combined with AI algorithms have shown better performance with the boom in artificial intelligence. These detection techniques are more accurate, robust and generalisable than traditional malware detection techniques, and can avoid the risk of false detection for many newly generated malware. Therefore, it is of better scientific interest to dig into malware detection systems based on this algorithms.

There are 2 main phases about malware detection techniques using artificial intelligence algorithms: the data pre-processing phase, which focuses on the extraction of software features, and the model classification phase, which uses the feature data to train the model to complete the classification task.

In the data pre-processing phase, the common extraction methods include static extraction and dynamic extraction about feature data. Static extraction of features means extracting features without running the software program [3]–[6], in ways that include extracting bytecode [7], file header information [8], API call information [9], application interface information [10], application permission information [8], etc. The main principle of static analysis features is to obtain the source code or bytecode of the program through software decompilation and analyze the semantic features and

The associate editor coordinating the review of this manuscript and approving it for publication was Marco Martalo.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

semantic information contained in it. The detection method using such features is included less overhead and stable, such as MaMadroid [11] proposed by Mariconti *et al.* and a malware detection method proposed by Wenjin Li *et al.* [12] which used Android-side application permission information, API call information and other static data for malware detection.

Compared with static extraction, the dynamic approach analyzes the behavioral activities of software runtime [13]–[18]. Therefore, the extracted features are more accurate, such as the DL-Droid proposed by Mohammed *et al.* [19], who used software log files running on real devices to extract feature data. They used more than 30,000 applications to extract feature data, and the accuracy was as high as 99.6%. Tobiyama *et al.* [20] used recurrent neural networks to extract feature from the temporal data of the processes when the malware program was running, and then used convolutional neural networks to classify them with a high accuracy of 96%. However, many malware programs hide their malicious behaviour in a virtual environment [21], and the dynamic virtual operating environment required to make them behave maliciously is more demanding and complex, so the classification model using such features is less stable in accuracy and more overheads.

In the model training and classification phase, the main approaches include malware detection methods based on machine learning algorithms and deep learning models. The methods based on machine learning algorithms mainly use common machine learning algorithms as classification models, like Wang *et al.* [22], used five machine learning models for software classification, namely Support Vector Machine (SVM), K-Nearest Neighbour (KNN), Naive Bayes (NB), Classification Regression Tree (CART) and Random Forest (RF). Kumar *et al.* [23] proposed a feature learning model using various machine learning algorithms to achieve detection of malware with low overhead and high accuracy. RepassDroid [24] extracted various APIs with sensitive trigger points and basic software permissions as datasets to train a machine learning model for detection. They used 24,288 samples for training and testing, and the experimental results show that their method had satisfactory results with accuracy rates of 97.7% and 93.3%, respectively. Yerima *et al.* [25] used a Bayesian classifier as classification model, and Li *et al.* [26] used a decision tree to construct a model to achieve classification and detection of malware.

Malware detection methods based on deep learning models mainly use neural networks [27], [28], recurrent neural networks and convolutional neural networks to implement malware detection. The malware detection methods applying recurrent neural network models are the most common. The methods based on this network structure usually encode all API instructions of the malware as one-hot vectors and put them into the model as input data, e.g. [29], [30]. Long-term short-term memory (LSTM) networks have shown on the metric of high accuracy [31]. However, RNNs are vulnerable to attacks. An attacker mimics the RNN used in the MDS based on inputs and outputs and adds redundant API calls using the adversarial RNN in an adversarial attack. Files injected using redundant API calls can easily bypass RNN detection [32]. Despite the high accuracy of RNNs, the reliability of the results generated by RNNs may still be questionable in malware detection. Convolutional neural networks can extract location non-specific local features from fixed-size, high-dimensional tensor-type data. As a result, it has been used and shown excellent performance in computer vision research, and there are also many research applications in the field of malware detection. Mahoud *et al.* [33] used 2D-CNN and 3D-CNN as classification models and used various detection data extracted from dynamic environments as feature data, with an accuracy of up to 90%. Xiao *et al.* [34] used CNN to understand the characteristics of Android malware from Dalvik bytecode. The method is efficient with an accuracy rate of over 93%. Wang *et al.* [35] proposed various network models for detecting malware, such as CNN-S, DAE-CNN-S, where malware data representation is extracted software privileged information to generate feature images, which outperforms most malware detection algorithms based on traditional machine learning models. Malware detection models using deep learning neural networks show superior detection performance, and are more scalable than malware detection models using machine learning algorithms.

In this work, we extract features from the bytecode of various command methods of android software in a static way. Then an auto-encoder based on convolutional neural network framework is used to reconstruct the grey-scale image corresponding to each malware. Finally, the auto-encoder is experimentally analysed in reconstructing the high-dimensional features of the malware performance. We designed a neural network based on the auto-encoder structure to perform the classification and detection task for malware. And experiments were conducted using datatsets we collected from VisureShare. The experimental results show that our method is more accurate than traditional machine learning methods and some deep learning malware detection models based on malware images.

The main contributions of this paper are as follows:

- We propose a method for generating feature images corresponding to each malware and benign software. The main approach is to convert the bytecodes of the various methods in the software into grey-scale images for subsequent model training and classification.
- We used auto-encoder based on convolutional neural network designed to recognise the high-dimensional features contained in such grey-scale images, and experimentally demonstrated the feasibility of the scheme.
- We propose a neural network model based on autoencoder networks for the classification task of malware detection and experimentally demonstrate the high accuracy of our malware detection model.

The remainder of this paper is summarised as shown follows: Section 2 presents the related work. We propose our theoretical scheme in section 3. Section 4 gives the experimental

results of malware detection model. Section 5 sumarises our liminations and section 6 summarises our work.

II. RELATED WORK

This section describes the work related to the generation of malware images and the static malware detection method based on deep learning models. Therefore, this section contains 2 parts, the first part is the malware image generation scheme and the second part is the static malware detection scheme based on deep learning models.

A. MALWARE IMAGES

In the feature extraction phase for malware detection, neural networks can also be used to extract the corresponding features of the software in addition to extracting the corresponding static feature information, such as API calls, permission information, etc, and dynamic feature information, such as network activity, log files, etc. This feature extraction solution is more automated and simpler than other manual feature extraction methods.

Automatic extraction of software features using neural nets requires consideration of the data representation. So that it can better extract the key features and ensure the accuracy of the test results. A feasibel solution is the use of images [36], where the program is transformed into an image and handed over to the neural net to extract the features. The similarity of software structures is reflected by the similarity of textures between corresponding images, such as the malware picture representation scheme proposed by Natarij *et al.* [37], they transformed the binary code of the malware into the form of a 2-dimensional matrix, and it can be represented in the form of a grey-scale graph since the numerical range of its transformed matrix is [0, 255], as shown in Fig. 1, where data of different structures have different textures.

Yan *et al.* [38] generated greyscale images from malware files while decompiling to obtain the software opcode sequences, trained the greyscale images using convolutional neural networks, learned the opcode sequences using long and short-term memory networks, and conducted experiments on more than 40,000 samples with an accuracy of 99.88%. They used bilinear interpolation to resize the images to ensure that the size of the greyscale images input to the training network should be the same size. K. He *et al.* [39] proposed a malware detection method based on image recognition. They converted malware into RGB images and classified them using CNN and spatial pyramid pooling (SPP) layer. Experimental evaluation showed that the malware detection method designed based on RGB images is highly accurate and resistant to redundant API injection attacks. ASLAN *et al.* [40] focused on the design of the network architecture of the detection model by converting PE files of software samples into grey-scale maps of malware, training and detecting them using a hybrid network structure, and testing them on the Malimg dataset with an accuracy of 97.98%. Nisa *et al.* [41] used distinctive pre-trained models (AlexNet and Inception-V3) for feature extraction, a hybrid model consisting of deep learning algorithms and traditional machine learning algorithms to achieve 99.3% accuracy for a 25-class malware classification task using data augmentation based on affine image transformation. Singh *et al.* [42] used 15 different combinations of Android malware image to identify and classify Android malware, and machine learning algorithms were used to analyse grey-scale malware images instead of the Softmax layer of CNN such as K-Nearest Neighbour (KNN), Support Vector Machine (SVM) and Random Forest (RF). The classification results showed that the method achieved a correct classification rate of 92.59%.

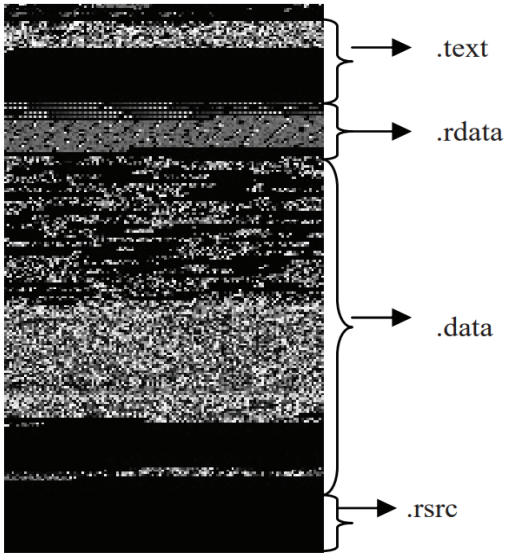

FIGURE 1. The malware grayscale image in [37].

These works focus on how to convert software of different sizes into images of the same size, and need to consider the challenge of how to do the best possible job of reducing redundancy in the process of generating the images. The difference between our work and previous work is that we try to extract the binary code of the method field in the software and convert some of the information into byte code to complete the generation of the grey-scale image. Analysing the feasibility of such a scheme is a major part of our research.

B. A STATIC MALWARE DETECTION SOLUTION BASED ON DEEP LEARNING MODELS

Deep learning models can show better performance on classification and prediction tasks [43], [44], so they have been widely used in many research areas [45], such as recommendation systems [46], privacy protection [47], [48], image recognition [49], and natural language processing. In the field of malware detection, deep learning models also have a wide range of applications [50].

The proposed malware detection scheme is related to the static feature extraction of software samples and the use of deep learning networks as classification detection models. Therefore, we present some noteworthy work in the area of malware detection models based on deep learning models. There are two reasons for choosing the static analysis

approach to extract software file features. Firstly, static analysis is intuitive and comprehensive, as compared to dynamic extraction efforts, static analysis does not need to consider when malware needs any trigger conditions to exhibit malicious behaviour, and its underlying source code intuitively contains the functional features of malware. Secondly, static analysis is faster and more efficient than dynamic detection, which takes a long time to run the malware program in order to record all kinds of data and is inefficient when dealing with a large number of software samples, whereas static analysis can extract features from a large number of software samples in a short period of time, which makes practical sense. The deep learning model was chosen because of its ability to generalise and detect previously unseen malware samples with high accuracy.

Wang *et al.* [6] obtained the corresponding manifest files and source code files from Android application files, extracted the corresponding software permission information and API function call information, then used deep learning algorithms to identify and classify them. The experiments proved that the proposed method has higher accuracy and stability compared with the traditional support vector machine method, and can identify similar features among similar malware. Yuan *et al.* [28] proposed a combination of static analysis methods and dynamic analysis methods for software feature extraction work in response to the current severe malware threat environment, statically analyzing the manifest configuration file and class execution file of Android software, and dynamically analyzing the log file of malware programs on the Android side, combining the two to extract software feature information. Then, they using deep learning models for training and classification, experimenting on more than 20,000 sample programs. The accuracy degree engaged to 96.76%. Kim *et al.* [27] conducted experiments on 41260 software samples, decompiled software executable files, extracted the corresponding software configuration files, execution files and function library files, used neural networks as classification models. And they proposed a multimodal deep neural network model by inputting different types of features into different initial neural networks for processing for features with different attributes. And finally the results are aggregated. The accuracy of the experimental data reached 98%. Li *et al.* [12] proposed a malware detection method based on weight-adjusted deep learning networks, which combined dangerous API calls and risky permission information as feature data, and the experimental results showed a high accuracy. These works focus on how to combine software feature data with deep learning models, using feature information that is not comprehensive enough and different from the feature data used in our work. Secondly, the deep learning models they use are relatively simple, and the robustness of their classification network detection accuracy for large-scale malware detection is controversial.

Shukla *et al.* [51] designed a malware detection model based on recurrent neural networks, using grey-scale images and hardware-based performance counters to extract feature, which improved the average accurate detection rate and precision by 11% compared to CNN-based sequence classification and Hidden Markov Model-based methods. The accuracy was as high as 94%. Chai *et al.* [52] obtained local semantic features from API call sequence information, learned them using cascaded convolutional neural networks and graph convolutional networks, proposed a joint framework for malware detection LGMal, and used the Alibaba Cloud Security malware detection dataset to conduct experiments with high accuracy. ZOU *et al.* [53] transformed the function call graph of a program into a complex social network and used the centrality analysis of social networks to perform the detection. The approach is to represent the semantic features of the graph by calculating the average closeness between sensitive API calls and the central node. Their detection method demonstrated 99.1% accuracy on 3988 benign and 4265 malicious samples, and was also six times faster than MaMaDroid. In our work, we try a different and novel idea to achieve detection. We use autoencoder network to design detection model, the design process of this network is complex but converges quickly and take less time to train.

III. APPROACH

A. OVERVIEW

We propose a approach to malware detection, which is designed based on the automatic encoder network. The Fig. 2 illustrates the overall structure and main tasks of our malware detection method. First, benign files and malware are transformed into corresponding greyscale images by decompiling the APK files, the binary codes are ectracted from methods in software, then converting them into decimal data by bytes, which are filled with pixel value. Afterwards, the greyscale images are passed through 2 deep learning networks in order to complete 2 tasks. The first deep learning network named automatic encoder network - 1(AE-1), which we use to analyse the feasibility of using grey-scale images to represent the corresponding features of softwares, and the second deep learning network is automatic encoder network - 2(AE-2), which we use to perform the task of classifying malicious softwares from benign softwares. The detailed design process of AE-1 and AE-2 will be described in the subsequent sections.

B. PRE-PROCESSING OF FEATURE DATA

The main task of the Pre-processing of feature data phase is to provide an input data for the neural network model. We use a grey-scale image of the software bytecode to represent the characteristics of the software, the so-called grey-scale image of the software bytecode is to decompile the software to obtain its binary bytecode, then convert it into a decimal type by byte and fill it into a fixed size two-dimensional matrix, since a byte is 8 bits, that corresponds exactly to the range of data from 0 to 255 and can be composed as a grey-scale image. The advantages of using this method are twofold. Firstly, this method of extracting software

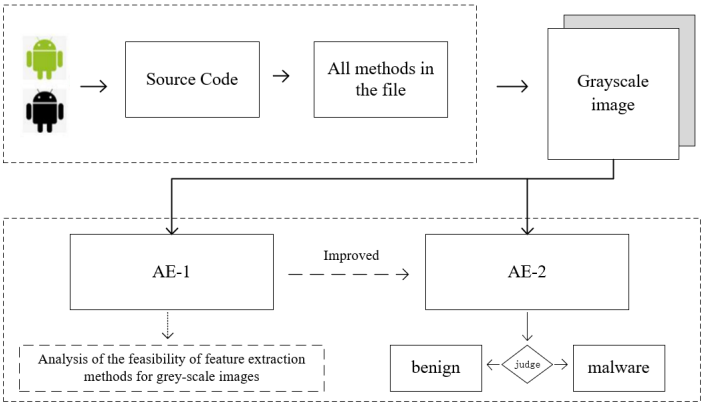

FIGURE 2. The overview of our proposed approach.

features is less overhead and intuitive. Secondly, the grey-scale image converted from the software file bytecode is a suitable input to the convolutional network for training and classification, as our subsequent network model is composed of a convolutional neural network, which requires a fixed size multi-dimensional matrix type of data.

However, the conversion of software binary codes into grey-scale images has some drawbacks. Although the software binary code contains a variety of feature, it also contains a large amount of works focus on how to convert software of different sizes into images of the same size, and need to consider the sticking points of how to do the best possible job of reducing redundancy in the process of generating the images. The difference between our work and previous work is that we try to extract the binary code of the method field in the software and convert some of the information into byte code to complete the generation of the grey-scale image. Analysing the feasibility of such a scheme is a major part of our research. The redundant information causes high pre-processing overhead and reduces the accuracy and robustness of the model classification at the later stage.

For this reason, we decompile the software and instead of converting the software binary data directly into a greyscale image. We extract all the methods in the software and convert the byte code of methods into a greyscale image, filling in any blank areas with zero. The advantage of this is twofold. Firstly, these methods contain various actions of the software, such as sending network data, reading private information on the phone, writing data to the phone's ROM and hard drive, and can be used to visually represent malicious actions in a greyscale image without setting up a dynamic runtime environment. The second point is that we have reduced the redundancy of using images to represent malware compared to previous grey-scale image processing, making the subsequent classification of the model more accurate and stable.

C. THE STRUCTURE OF OUR AUTOENCODER

The autoencoder network structure is a special kind of unsupervised neural network in a deep learning model [54]. It consists of an encoding network and a decoding network, as shown in Fig. 3. The encoding network achieves the effect of dimensionality reduction and compression, and the decoding network achieves the purpose of reconstructing the input. Its loss function is defined as the error value between the original input and the model output corresponding to the original input, and minimising its loss function by means of training and gradient updating is the operation process of the autoencoder network. Borghesi *et al.* used autoencoders to enable anomaly detection in large computer systems [55]. Their results show that the autoencoder can monitor anomalies that were never noticed before based on previous log records with an accuracy of between 88% and 96%. Angelo *et al.* propose a malware detection system for Android based on an autoencoding network [56]. They put sequences of API calls from the application as input into an autoencoder network to complete feature extraction, then used a neural network to train and classify features. Their system achieves higher accuracy than complex traditional machine learning methods such as J48, Naive Bayes and MLP.

We designed 2 model structures and named them AE-1 and AE-2 respectively, the design sequence is AE-1 first and then AE-2. The main purpose of designing the AE-1 network is to use it to analyse the feasibility of feature extraction methods for grey-scale images, and the purpose of designing the AE-2 network is to use it for malware detection. The reason for designing the 2 networks is that the AE-1 network exhibited more drawbacks and less stability for the experimental aspects of the classification task, so we improved on the AE-1 network and proposed the AE-2 network. It is worth noting that the AE-1 network is trained in an unsupervised manner and no software samples are labelled, while the AE-2 network is trained in a supervised manner and requires labelling of malicious and benign software samples. The specific structure of two networks will be described in the subsequent Part I and Part II.

1) THE FIRST AUTOMATIC ENCODER STRUCTURE (AE-1)

The structure of model AE-1 is shown in Fig. 4, and consists of convolutional layers, pooling layers and

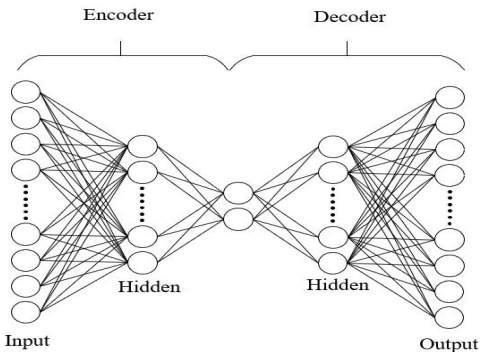

FIGURE 3. The schematic representation of an autoencoder.

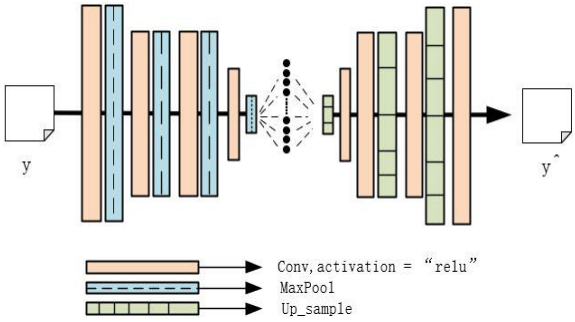

FIGURE 4. The first automatic encoder structure (AE-1).

up-sampling layers. The activation function uses the relu function and the MAE loss function with the Equ.(1):

$$l_r = \frac{1}{2n}\sum_i (y_i - y_i^r)^2 \tag{1}$$

Model AE-1 analyzes whether it can reconstruct the malware feature image based on the magnitude of similarity between the original malware feature image and the reconstructed image that has been reconstructed through the autoencoder network.

We determine whether AE-1 is able to perform this task by analyzing the numerical magnitude of its similarity, and this measure of similarity is expressed by defining the *SimiliarError* as the following Equ.(2).

$$SimiliarError = \frac{\sum_{i=0}^{N}(|y_r^i - y_g^i|)}{N} \tag{2}$$

where y_r^i is a pixel of the original image, y_g^i is the pixel corresponding to the original position in the image generated by the autoencoder, and N is all the pixel points of an image.

The theoretical basis for determining whether AE-1 can perform this task based on the numerical magnitude of the *SimiliarError* is that only unlabeled malware datasets are employed during the training phase of the autoencoder network. In the predictive classification phase, if a feature image corresponds to a category that is malware, then the reconstructed image it generates via the autoencoder network is the same as the original. The similarity between images will be high and the *SimiliarError* value will be decreased, because this is what the autoencoder network is trained on. On the contrary, if a feature image corresponds to a category belonging to benign software, the similarity between the reconstructed image generated by model and the original image will be low and the *SimiliarError* value will be enlarged, since the structure of a feature image transformed by benign software is very different from the structure of a feature image transformed by malware. For example, if we ask an expert in real life to focus on malware without studying the characteristics of benign software, he can easily distinguish the difference between benign and malware if he is knowledgeable in the high-dimensional characteristics of malware and then looks at benign software. If the error value of *SimiliarError* generated by the two types of software after such an autoencoder have a huge difference, then we can use this to make sure that the autoencoder network can indeed reconstruct the corresponding feature images of the two types of software better.

2) THE SECOND AUTOMATIC ENCODER STRUCTURE (AE-2)

The structure of model AE-2 is shown in Fig. 5, in which the autoencoder network structure is similar to model AE-1. The only difference is that we have an external multi-layer perceptron network to facilitate classification and experimental evaluation. We first extract the high-dimensional features corresponding to malware and benign software from model AE-1 by pre-training, then extract the output from the hidden layer of model AE-1 and use it for the training of the multilayer perceptron network. The multilayer perceptron network outputs two-dimensional vectors to complete the malware and benign software classification task.

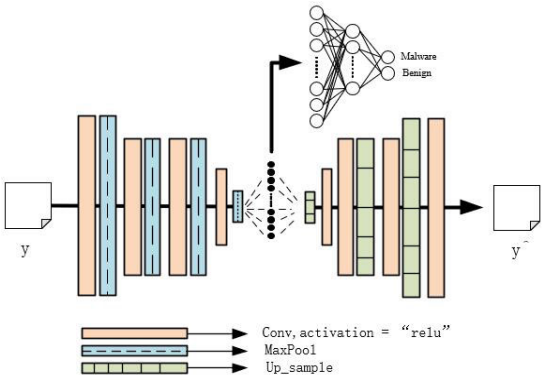

FIGURE 5. The second automatic encoder structure (AE-2).

IV. EXPERIMENT EVALUATION

In this section, we evaluate the proposed approach through experiments under different indicators. This section includes four parts, namely experimental setup, data feature extraction, effectiveness analysis of reconstructing malware images and performance analysis of the detection model.

A. EXPERIMENTAL SETUP

This subsection focuses on information related to the experimental setup and, for this purpose, is divided into 3 sections,

TABLE 1. Experimental environment setup.

Experimental environment setup	Requirements
CPU	Intel CoreTM i5-8300
RAM	16GB RAM
GPU	GeForce GTX 1060 MQ
Operating System	64-bit windows10 operating system
Programming Frameworks	Tensorflow 2.1, and Python 3.7

namely the experimental environment setup, the dataset, and the training details.

1) EXPERIMENTAL ENVIRONMENT SETUP

The details are shown in table 1 for information on the experimental environment. Our experiments were conducted using an Intel CoreTM i5-8300 machine with 16GB RAM, and GeForce GTX 1060 MQ. The machine had a 64-bit windows10 operating system. We used Keras, Tensorflow 2.1, and Python 3.7 for programing purposes.

2) DATASET

To evaluate the performance of the proposed model, we collected benign software from the Google App Store [57] and malware from VirusShare [58], where benign software consisted of 10 categories such as office, video, gaming, finance, photography and reading, and malware included datasets for APK categories released in 2016, 2017 and 2018. VirusTotal [59] scans a random sample of software to determine that they are correctly labled.

We divided them into 3 types of datasets according to their purpose, namely: (1) Dataset-1, this dataset is used for training and evaluation of AE-1 models, which includes 8121 malware and 2000 benign softwares. (2) Dataset-2, this dataset is used for training, validation and testing of the AE-2 model and contains 8121 malware and 7015 benign software. (3) Dataset-3, this dataset is used to analyse the detection performance of the AE-2 model on unseen software and includes 5,384 malware and 5,000 benign software. It is worth noting that when we divided Dataset-2 and Dataset-3, we deliberately put older software samples into Dataset-2 for training, e.g. malware from 2016, and newer releases into Dataset-3, e.g. 2017, 2018. The purpose of this is to simulate the scenario when the model detects new softwore samples released in the future and to faciliate the analysis of its performance.

3) TRAIN AND TEST DEAILS

The AE-1 network is used for the task of analyzing the performance of the autoencoder to reconstruct feature images, and detailed parameters of model AE-1 are shown in Table 2. The Adam optimization algorithm is used in the training phase and we set the learning learning rate to be 1e-4, the epoch is 100.

We divide the dataset-1 into 3 parts, the training dataset D_{Train}, which contains a partial dataset of the collected malware software, the malware test set D_{Test_mal}, which contains a partial dataset of the collected malware, and the benign software test set D_{Test_benign}, which contains a dataset of the collected benign software files.

TABLE 2. Parameters of AE-1 model.

Layer	Output shape	Parameters
$input$	[(None,500,500,1)]	0
$maxpooling2d$	[(None,250,250,1)]	0
$conv\_2d$	[(None,250,250,8)]	80
$maxpooling2d$	[(None,125,125,8)]	0
$conv\_2d$	[(None,125,125,8)]	584
$maxpooling2d$	[(None,63,63,8)]	0
$conv\_2d$	[(None,63,63,6)]	584
$up\_sampling2d$	[(None,126,126,8)]	0
$conv\_2d$	[(None,126,126,8)]	584
$up\_sampling2d$	[(None,252,252,8)]	0
$conv_2d$	[(None,250,250,16)]	1168
$up\_sampling2d$	[(None,500,500,16)]	0
$conv\_2d$	[(None,500,500,1)]	145

The AE-1 network use training dataset D_{Train} for the training task, then use the malware test set D_{Test_mal} and the benign software test set D_{Test_benign} for the test task. If the new input of test set is similar to the input of the dataset used in the training phase, then the reconstruction error for this test set is very small. Conversely, if the new inputs of test set are different from the inputs to the dataset used in the training phase, then this test set will exhibit a very large reconstruction error. The large difference in the error data produced by these 2 test sets after AE-1 is exactly what we are experimenting with. Since our hypothesis is based on the theory that malware is all similar and benign software is not similar to malware, in practice, the different functional characteristics exhibited between malware families in the malware dataset and the large redundancy characteristics contained in the software dataset can lead to experimental results exhibiting large instabilities. For this reason, we are more interested in the relative differences between the 2 test sets than in the absolute errors they exhibit.

The AE-2 network is used for the task of analyzing the performance of the detection model. For the overall dataset partitioning, we used 80% of the Dataset-2 as the training set and 20% as the test set. In the training phase, the training set was trained and validated using k-fold cross-validation, with k = 6, meaning that 5/6 of the training set was used for training and 1/6 for validation, repeated 6 times, and finally the average was taken. In the testing phase, the test set is used for testing. Minutes are used as units for training time. The Adam optimization algorithm, learning rate of 0.0001 and epoch of 100 are chosed in AE-2's training.

The variety of evaluation metrics such as FPR, TPR, ACC, Precision and F-score are used in the model evaluation test phase, which are calculated as shown below.

$$FPR = FP/(FP + TP) \tag{3}$$

$$TPR = Recall = TP/(TP + FN) \tag{4}$$

$$Acc = (TP + TN)/(TP + TN + FP + FN) \quad (5)$$

$$Precision = TP/(TP + FP) \quad (6)$$

$$F1 - score = 2 \cdot Precision \cdot Recall/(Precision + Recall) \quad (7)$$

B. DATA FEATURE EXTRACTION

We used the Androguard tool to complete the data pre-processing task of the model, extracting the source code of all the class files in the APK file through the Androguard analysis framework, extracting the bytecodes of all the methods and converting them into the decimal data needed for the corresponding grey-scale images. In the sample dataset of software collected, we chose files with as small a data size as possible to ensure that we could standardise the size of all images.

Based on this approach, all software is converted into the feature images we need during the data pre-processing phase.

C. EFFECTIVENESS ANALYSIS OF RECONSTRUCTING MALWARE IMAGES

In this subsection, we evaluate the effectiveness of reconstructing malware images by analysing the overall error distribution in malware and benign reconstruct malware images. Fig. 6 shows the overall error distribution for the 2 test sets. The reconstructed error value generated by each software after the encoder network are normalised and expressed as value on the Y-axis. We normalise by adding up the error value for each pixel point corresponding to the feature image of the malware and dividing by the total. In the line statistics graph, the blue line represents the error trend for the overall D_{Test_mal} and the yellow line represents the error trend for the overall D_{Test_benign}. The error is not exactly zero due to the inherent variability contained in the dataset and the redundancy of the software files. However, as can be seen in Fig. 6, the overall error trend is stable for the malware dataset represented by the blue line, whereas the overall error trend for the benign software test set represented by the yellow line is unstable and fluctuates widely, and the relative difference between the mean value of the errors presented by the two datasets is large. Thus, our theory is plausible.

We conducted a quantitative analysis of the two test sets and normalised the value and presented them in Table 3, where the normalisation was done by calculating the mean absolute error (MAE) and root mean square error (RMSE) of the test set after making the error of the training set equal to 1. The experimental data, as described in Table 3, showed that the normalised MAE and normalised RMSE produced by the malware test set were close to 1, indicating that the D_{Test_mal} was similar to the D_{Train}, while the benign test set produced a normalised MAE and normalised RMSE greater than 1 and also greater than the value of the training set, indicating that the benign software test set was not similar to the training set. The normalised MAE and normalised RMSE for the software test set were greater than 1 and also greater than the value for the malware test set, indicating that the D_{Test_benign} was not similar to the D_{Train}, and this quantitative analysis also demonstrated that the network structure would show similar results on the invisible data set.

TABLE 3. Quantitative analysis of two datasets.

datasets	D_{Test_mal}	D_{Test_benign}
Normalized MAE	1.12	2.02
Normalized	1.25	2.89

Based on this experiment, we can then show that the task of reconstruction can be performed well by the automatic encoder through the pre-processed malware data from our data, and that the automatic encoder can identify high-dimensional features of both benign and malicious software. Then, we implement the subsequent task of classifying malware and benign software.

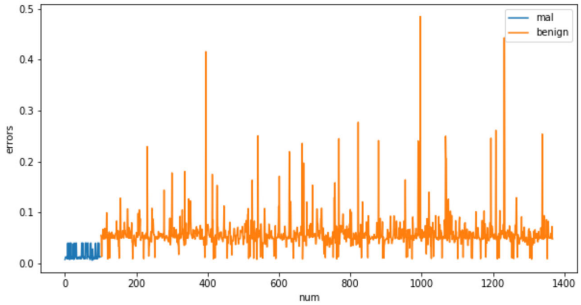

FIGURE 6. Reconstruction error for two datasets.

D. PERFORMANCE ANALYSIS OF THE DETECTION MODEL

We use the AE-2 model to analyse the classification performance of autoencoders in this subsection. To the end, we experimented with some similar previous research work for comparative analysis, include detection models using traditional machine learning algorithms [22], detection models designed based on recurrent neural network [29], detection models designed based on autoencoder networks and convolutional neural networks [35] and detection models using Malware Images [39]. Among them, the detection model CNN-SPP designed with malware images and convolutional neural networks proposed by He and Kim [39] showed high accuracy on the dataset provided by Seoul National University [60]. The DAE-CNN model proposed by Wang *et al.* [35] uses an autoencoder network as the data preprocessing model and the output data of the model is used for training and detection of convolutional neural networks, which is similar to our theoretical model, and they show better results on 10,000 benign APPs and 13,000 malicious APPs. We use these 2 types of models as baseline models for comparison experiment with AE-2 on different datasets.

1) PERFORMANCE COMPARISON OF DIFFERENT MODELS

The ROC curves in Fig. 7 show the effect of the model on the training set, from which it can be seen that the model exhibits a more stable performance on the training set. The ROC curves in Fig. 8 show the model's performance on the test set

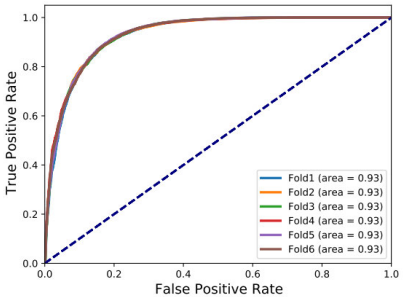

FIGURE 7. The ROC curve of AE-2 on training set.

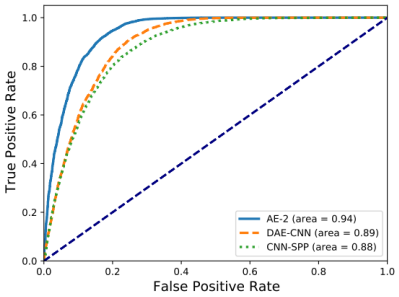

FIGURE 8. The ROC curve of different models on the test set.

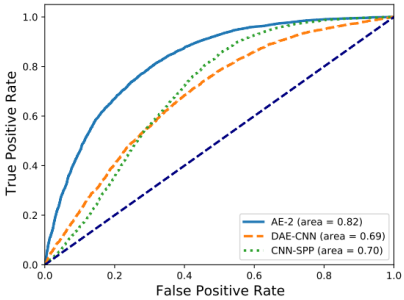

FIGURE 9. The ROC curve of different models on the unseen software.

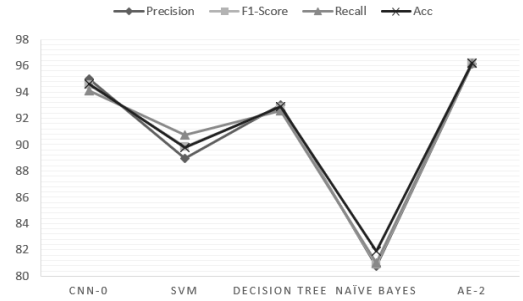

FIGURE 10. Comparation results in five different models.

which in Dataset-2. We can see that our model outperforms the other two.

To further examine the detection performance of our model on unseen malware, the Datasets-3 is used as test set for AE-2. The ROC curves are shown in Fig. 9, from which it can be seen that our model shows good accuracy and some feasibility in detecting unseen malware, but also shows some flaws as the software changes with year iteration.

In the experimental results presented in Table 4, the high F1-score value for the AE-2 model demonstrate the feasibility and stability of our solution. The traditional machine learning algorithms and autoencoder have higher F-scores due to the use of multiple fine-grained feature extraction methods, and the efficient feature extraction method is the key to determine the performance of the model. The lower F1-score shown in recurrent neural network model is due to the fact that its feature extraction method is complex and the model built using RNN shows unstable performance on the dataset. The lower F1-score in convolutional neural network model demonstated that this way of constructing feature images based on file binary codes contains more redundant information and less distinct features.

2) DETAILED COMPARISON OF MULTIPLE INDICATORS

After that, we use a variety of common machine learning models and deep learning models to conduct experiments comparing various evaluation metrics. The common machine learning algorithms include support vector machines, decision trees and naive bayes, and the deep learning model named CNN-0, which model consists of one layer of convolution, one layer of pooling and one layer of fully connected neural network, we use it as our benchmark model. Fig. 10 illustrates the value of accuracy, precision, recall and F-score on the five models. We find that the decision tree outperforms the support vector machine model and the naive Bayes model in terms of accuracy and performance for traditional machine learning detection algorithms through cross-sectional comparisons, while the deep learning model outperforms the traditional machine learning algorithm in terms of overall performance. Our model achieves better results in terms of search accuracy and completeness.

Fig. 11 illustrates the performance comparison between the two different deep learning models. It can be seen from the figure that AE-2 spends less training time compared to the CNN-0 model, and the ACC, recall, Precision, and F-score metrics are all very similar. For FPR, AE-2 shows lower value.

Table 5 shows all the experimental data. The training time for our model is 1407.32 mins, which is about 23.45 hours. For the CNN-0 model, the number of parameters in the image data is huge and it takes a significant amount of time, 28.14 hours, after performing the convolution and pooling operations since the structure contains only one layer of convolution and one layer of pooling.

V. LIMINATION

There are 2 main limitations of our model. The first point is that the data pre-processing method needs to be improved. Although we did our best to reduce the redundant information carried by the software feature representation, there is still the problem of inefficiency, and the dataset we used is smaller due to the limitations of our experimental environment. The feasibility and effectiveness of this approach on other

TABLE 4. Performance comparison of different models.

Study,Year	Alogrithm / Model	Features	F1-Score
[22],2018	traditional machine learning algorithms	11 types of static features from each app	92.78
[29],2015	recurrent neural network (RNN)	malware event stream	83.56
[39],2019	convolutional neural networks (CNN)	converts binary malwares files to grayscale/RGB images	86.27
[35],2019	autoencoders	API call sequence	94.69
AE-2	autoencoders and fully connected neural network	the byte code of software methods	96.17

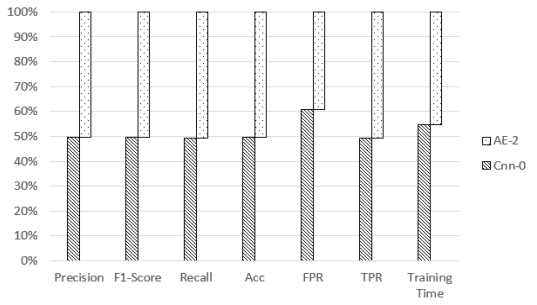

FIGURE 11. Performance comparison of 2 deep learning models.

TABLE 5. Detailed comparison of multiple indicators.

Classifier	FPR(%)	TPR(%)	ACC(%)	Precision(%)	Recall(%)	F1-Score(%)	Traning time(mins)
CNN-0	5.90	94.10	94.60	94.96	94.10	94.53	1688.54
SVM	11.09	90.70	89.79	88.96	90.70	89.81	1.86
Decision Tree	8.79	92.59	92.90	93.03	92.60	92.81	0.62
Naive Bayes	18.99	81.01	81.88	80.78	81.01	80.89	1.575
AE-2	3.80	96.20	96.22	96.14	96.20	96.17	1407.32

types of malware dataset will require more detailed analysis and research in the future. The second point is that the instability of detection performance caused by deep learning models is difficult to estimate. Although deep learning algorithms, such as convolutional neural networks, have a promising future in areas such as image recognition and text generation, the use of malware feature data for classification tasks in deep learning models can lead to unstable detection performance, because the models are very dependent on the original training dataset, and the higher the accuracy, the greater the dependency, and the detection accuracy for software samples that are not in the training set will be reduced.

VI. CONCLUSION

In this paper, we propose a novel approach to malware detection, which is based on the principle of using grey-scale images to represent the features of malware and using an auto-encoder network to design a classification model to achieve malware detection. Experimental results show the feasibility of our proposed approach of converting the bytecode of all methods in software into a greyscale image to represent the features in a software sample. Compared to malware detection methods designed based on traditional machine learning algorithms, our method is more accurate. Our method requires less training time and detection time compared to other malware detection systems designed based on deep learning models. In future work, we will continue to explore more effective methods for representing malware feature images and focus our research on the data pre-processing stage to explore newer malware detection methods.

ACKNOWLEDGMENT

An earlier version of this paper was presented at the IEEE MASS2020, Delhi, India, October 1-4, 2020 [DOI: 10.1109/MASS50613.2020.00009], and expanded version of a paper entitled 'A Malware Detection Approach Using Malware Images and Autoencoders.'

REFERENCES

[1] (2019). *China Internet Security Research Report*. (Nov. 15, 2020). [Online]. Available: https://www.cert.org.cn/publish/main/upload/File/2019Annual%20report.pdf

[2] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 1–40, May 2018.

[3] S. Rastogi, K. Bhushan, and B. B. Gupta, "Android applications repackaging detection techniques for smartphone devices," *Proc. Comput. Sci.*, vol. 78, pp. 26–32, Jan. 2016.

[4] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "WHYPER: Towards automating risk assessment of mobile applications," in *Proc. 22nd USENIX Secur. Symp. (USENIX Security)*, 2013, pp. 527–542.

[5] W. Klieber, L. Flynn, A. Bhosale, L. Jia, and L. Bauer, "Android taint flow analysis for app sets," in *Proc. 3rd ACM SIGPLAN Int. Workshop State Art Java Program Anal. (SOAP)*, 2014, pp. 1–6.

[6] Z. Wang, J. Cai, S. Cheng, and W. Li, "DroidDeepLearner: Identifying Android malware using deep learning," in *Proc. IEEE 37th Sarnoff Symp.*, Sep. 2016, pp. 160–165, doi: 10.1109/SARNOF.2016.7846747.

[7] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proc. IEEE Symp. Secur. Privacy. (S&P)*, May 2001, p. 2001, doi: 10.1109/SECPRI.2001.924286.

[8] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Android permissions: A perspective combining risks and benefits," in *Proc. 17th ACM Symp. Access Control Models Technol. (SACMAT)*, 2012, pp. 13–22.

[9] C. Zhao, W. Zheng, L. Gong, M. Zhang, and C. Wang, "Quick and accurate Android malware detection based on sensitive Apis," in *Proc. IEEE Int. Conf. Smart Internet Things (SmartIoT)*, Aug. 2018, pp. 143–148.

[10] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "ANASTASIA: ANdroid mAlware detection using STatic analySIs of applications," in *Proc. 8th IFIP Int. Conf. New Technol., Mobility Secur. (NTMS)*, Nov. 2016, pp. 1–5.

[11] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android malware by building Markov chains of behavioral models," 2016, *arXiv:1612.04433*.

[12] W. Li, Z. Wang, J. Cai, and S. Cheng, "An Android malware detection approach using weight-adjusted deep learning," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Mar. 2018, pp. 437–441, doi: 10.1109/ICCNC.2018.8390391.

[13] B. Amos, H. Turner, and J. White, "Applying machine learning classifiers to dynamic Android malware detection at scale," in *Proc. 9th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jul. 2013, pp. 1666–1671.

[14] S. Nari and A. A. Ghorbani, "Automated malware classification based on network behavior," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Jan. 2013, pp. 642–647.

[15] G. Cabau, M. Buhu, and C. P. Oprisa, "Malware classification based on dynamic behavior," in *Proc. 18th Int. Symp. Symbolic Numeric Algorithms Sci. Comput. (SYNASC)*, Sep. 2016, pp. 315–318.

[16] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Trans. Comput. Syst.*, vol. 32, no. 2, pp. 1–29, Jun. 2014.

[17] W.-C. Wu and S.-H. Hung, "DroidDolphin: A dynamic Android malware detection framework using big data and machine learning," in *Proc. Conf. Res. Adapt. Convergent Syst. (RACS)*, 2014, pp. 247–252.
[18] 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.
[19] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DL-droid: Deep learning based Android malware detection using real devices," *Comput. Secur.*, vol. 89, Feb. 2020, Art. no. 101663.
[20] 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.
[21] D. Shi, X. Tang, and Z. Ye, "Detecting environment-sensitive malware based on taint analysis," in *Proc. 8th IEEE Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Nov. 2017, pp. 322–327, doi: 10.1109/ICSESS.2017.8342924.
[22] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, "Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers," *Future Gener. Comput. Syst.*, vol. 78, pp. 987–994, Jan. 2018.
[23] A. Kumar, K. S. Kuppusamy, and G. Aghila, "A learning model to detect maliciousness of portable executable using integrated feature set," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 31, no. 2, pp. 252–265, Apr. 2019.
[24] N. Xie, F. Zeng, X. Qin, Y. Zhang, M. Zhou, and C. Lv, "RepassDroid: Automatic detection of Android malware based on essential permissions and semantic features of sensitive Apis," in *Proc. Int. Symp. Theor. Aspects Softw. Eng. (TASE)*, Guangzhou, China, Aug. 2018, pp. 52–59.
[25] S. Y. Yerima, S. Sezer, and G. McWilliams, "Analysis of Bayesian classification-based approaches for Android malware detection," *IET Inf. Secur.*, vol. 8, no. 1, pp. 25–36, Jan. 2014.
[26] Q. Li and X. Li, "Android malware detection based on static analysis of characteristic tree," in *Proc. Int. Conf. Cyber-Enabled Distrib. Comput. Knowl. Discovery*, Sep. 2015, pp. 84–91.
[27] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for Android malware detection using various features," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 3, pp. 773–788, Aug. 2019, doi: 10.1109/TIFS.2018.2866319.
[28] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: Android malware characterization and detection using deep learning," *Tsinghua Sci. Technol.*, vol. 21, no. 1, pp. 114–123, Feb. 2016, doi: 10.1109/TST.2016.7399288.
[29] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, South Brisbane, QLD, Australia, Apr. 2015, pp. 1916–1920.
[30] X. Wang and S. M. Yiu, "A multi-task learning model for malware classification with useful file access pattern from API call sequence," 2016, *arXiv:1610.05945*.
[31] B. Athiwaratkun and J. W. Stokes, "Malware classification with LSTM and GRU language models and a character-level CNN," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, New Orleans, LA, USA, Mar. 2017, pp. 2482–2486.
[32] W. Hu and Y. Tan, "Black-box attacks against RNN based malware detection algorithms," in *Proc. Workshops 32nd AAAI Conf. Artif. Intell.* New Orleans, LA, USA: AAAI Press, 2018, pp. 245–251.
[33] M. Abdelsalam, R. Krishnan, Y. Huang, and R. Sandhu, "Malware detection in cloud infrastructures using convolutional neural networks," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2018, pp. 162–169, doi: 10.1109/CLOUD.2018.00028.
[34] X. Xiao, "An image-inspired and CNN-based Android malware detection approach," in *Proc. 4th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, San Diego, CA, USA, Nov. 2019, pp. 1259–1261.
[35] W. Wang, M. Zhao, and J. Wang, "Effective Android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *J. Ambient Intell. Hum. Comput.*, vol. 10, no. 8, pp. 3035–3043, Aug. 2019.
[36] T. H.-D. Huang and H.-Y. Kao, "R2-d2: ColoR-inspired convolutional NeuRal network (CNN)-based AndroiD malware detections," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 2633–2642.
[37] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. 8th Int. Symp. Visualization Cyber Secur.*, 2011, pp. 1–7.
[38] J. Yan, Y. Qi, and Q. Rao, "Detecting malware with an ensemble method based on deep neural network," *Secur. Commun. Netw.*, vol. 2018, pp. 1–16, 2018.
[39] K. He and D.-S. Kim, "Malware detection with malware images using deep learning techniques," in *Proc. 18th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./13th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Aug. 2019, pp. 95–102.
[40] O. Aslan and A. A. Yilmaz, "A new malware classification framework based on deep learning algorithms," *IEEE Access*, vol. 9, pp. 87936–87951, 2021.
[41] M. Nisa, J. H. Shah, S. Kanwal, M. Raza, M. A. Khan, R. Damaševičius, and T. Blažauskas, "Hybrid malware classification method using segmentation-based fractal texture analysis and deep convolution neural network features," *Appl. Sci.*, vol. 10, no. 14, p. 4966, Jul. 2020, doi: 10.3390/app10144966.
[42] J. Singh, D. Thakur, F. Ali, T. Gera, and K. S. Kwak, "Deep feature extraction and classification of Android malware images," *Sensors*, vol. 20, no. 24, p. 7013, Dec. 2020, doi: 10.3390/s20247013.
[43] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[44] Y. Bengio, "Learning deep architectures for AI," *Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.
[45] X. Yan, Y. Xu, X. Xing, B. Cui, Z. Guo, and T. Guo, "Trustworthy network anomaly detection based on an adaptive learning rate and momentum in IIoT," *IEEE Trans. Ind. Informat.*, vol. 16, no. 9, pp. 6182–6192, Sep. 2020.
[46] X. Li, W. Jiang, W. Chen, J. Wu, G. Wang, and K. Li, "Directional and explainable serendipity recommendation," in *Proc. Web Conf.*, Apr. 2020, pp. 122–132.
[47] H. Elahi, G. Wang, T. Peng, and J. Chen, "AI and its risks in Android smartphones: A case of Google smart assistant," in *Dependability in Sensor, Cloud, and Big Data Systems and Applications*, vol. 1123, G. Wang, M. Z. A. Bhuiyan, S. D. C. di Vimercati, and Y. Ren, Eds. Guangzhou, China: Springer, Nov. 2019, pp. 341–355.
[48] Y. Xu, G. Wang, J. Ren, and Y. Zhang, "An adaptive and configurable protection framework against Android privilege escalation threats," *Future Gener. Comput. Syst.*, vol. 92, pp. 210–224, Mar. 2019.
[49] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[50] R. Nix and J. Zhang, "Classification of Android apps and malware using deep neural networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 1871–1878.
[51] S. Shukla, G. Kolhe, S. M. Pd, and S. Rafatirad, "RNN-based classifier to detect stealthy malware using localized features and complex symbolic sequence," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2019, pp. 406–409, doi: 10.1109/ICMLA.2019.00076.
[52] Y. Chai, J. Qiu, S. Su, C. Zhu, L. Yin, and Z. Tian, "LGMal: A joint framework based on local and global features for malware detection," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, Jun. 2020, pp. 463–468, doi: 10.1109/IWCMC48107.2020.9148289.
[53] D. Zou, Y. Wu, S. Yang, A. Chauhan, W. Yang, J. Zhong, S. Dou, and H. Jin, "IntDroid: Android malware detection based on API intimacy analysis," *ACM Trans. Softw. Eng. Methodology*, vol. 30, no. 3, pp. 1–32, May 2021, doi: 10.1145/3442588.
[54] (Jan. 15, 2020). *The Keras Blog*. [Online]. Available: https://blog.keras.io/building-autoencoders-in-keras.html
[55] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Anomaly detection using autoencoders in high performance computing systems," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 9428–9433.
[56] G. D'Angelo, M. Ficco, and F. Palmieri, "Malware detection in mobile environments based on autoencoders and API-images," *J. Parallel Distrib. Comput.*, vol. 137, pp. 26–33, Mar. 2020.
[57] (Jan. 15, 2020). *Google Play Store*. [Online]. Available: https://developer.android.google.cn/distribute/google-play
[58] (Jan. 15, 2020). *Virusshare*. [Online]. Available: http://virusshare.com/
[59] (Jan. 15, 2020). *Virustotal*. [Online]. Available: https://www.virustotal.com/ko
[60] J.-W. Jang, H. Kang, J. Woo, A. Mohaisen, and H. K. Kim, "Andro-dumpsys: Anti-malware system based on the similarity of malware creator and malware centric information," *Comput. Secur.*, vol. 58, pp. 125–138, May 2016.