Deep Image: A precious image-based deep learning method for online malware detection in IoT Environment

Meysam Ghahramani, Rahim Taheri, Member, IEEE, Mohammad Shojafar, Senior Member, IEEE, Reza Javidan, Shaohua Wan, Senior Member, IEEE

Abstract—The volume of malware and the number of attacks in IoT devices are rising everyday, which encourages security professionals to continually enhance their malware analysis tools. Researchers in the field of cyber security have extensively explored the usage of sophisticated analytics and the efficiency of malware detection. With the introduction of new malware kinds and attack routes, security experts confront considerable challenges in developing efficient malware detection and analysis solutions. In this paper, a different view of malware analysis is considered and the risk level of each sample feature is computed, and based on that the risk level of that sample is calculated. In this way, a criterion is introduced that is used together with accuracy and FPR criteria for malware analysis in IoT environment. In this paper, three malware detection methods based on visualization techniques called the clustering approach, the probabilistic approach, and the deep learning approach are proposed. Then, in addition to the usual machine learning criteria namely accuracy and FPR, a proposed criterion based on the risk of samples has also been used for comparison, with the results showing that the deep learning approach performed better in detecting malware.

Index Terms—Malware Detection, Image-based Clustering, Deep Learning, IoT Devices, Visualization Analysis.

I. INTRODUCTION

In today’s digital world, with the rise of IoT devices malware development continues at a dizzying rate, despite measures to recognise and combat it. Malware analysis is essential to defend against the sophisticated behaviour of this form of malware. Manual heuristic inspection, on the other hand, is no longer effective or efficient. To address these basic challenges, methods based on behavior-based malware detection mixed with machine learning approaches have been enthusiastically accepted. It applies supervised classifiers to assess their forecasting ability in terms of identifying the most relevant features from the original feature set and balancing a high detection rate with cheap computation cost. While machine learning-based malware detection systems have proved efficient at identifying malware, their shallow learning architecture is still not good enough for complicated malware to be identified.

The exponential growth of malware is owing to the financial rewards connected with malware vulnerabilities such as cryptocurrency coin miners, banking Trojans, and ransomware. Analysts depend largely on machine learning algorithms to discover and classify malware intelligently. Machine learning algorithms will develop models of benign software and malware in order to identify malware in IoT environments. The features that are selected and made available are crucial to the effective utilization of these automation systems. These aspects comprise data acquired from static binary executable files and dynamic malware/software execution behavior.

The majority of deep learning-based windows malware detection algorithms now identify the test input file using static PE file attributes. They also create an image from the whole content of the PE file. Because PE files come in a variety of sizes, a scaling technique is necessary to transform images with different dimensions to a single dimension. The static feature-based detection and classification technique’s main drawback is that it is easily evaded by disguised malware. However, most of the present research in malware analysis is based on the assumption that a malware sample is accessible. Frequently, this is not the case. For instance, advanced persistent threat organisations are known to conduct targeted attacks and then self-delete the malware they have planted.

The methods used in most malware detection systems are focused on increasing accuracy and decreasing FPR. However, in this paper, we show that although these two metrics are very suitable, they are not enough and in some cases even with high accuracy and low FPR, the detection system does not work properly. Therefore, in this paper, a metric is presented that can be used in addition to accuracy and FPR to improve the detection system. As will be demonstrated, in contrast to the research literature, which defines accuracy and FPR in terms of reducing mean squared error, the strategy described in this paper is risk-based. This implies that for each sample, a risk percentage is considered, which is a value between zero and one, and the risk criterion is set and a decision is made using this risk percentage.
A method based on deep learning is proposed that has been shown in eq. [1]. In this representation, $0 \leq \xi \leq 1$ shows the maliciousness level. Also, each malware has features $\phi_i$'s, $1 \leq i \leq n$, so that the feature $\phi_i$ is available at the time $\tau_i$.

$$\Xi = [\xi, \phi_1: \tau_1, \phi_2: \tau_2, \cdots, \phi_n: \tau_n]$$  \hspace{1cm} (1)

This paper intends to propose a malware detection method to calculate $\xi$ from $[\phi_1: \tau_1, \cdots, \phi_n: \tau_n]$. Note that eq. [1] is not the only representation for $\Xi$. For example, a two-dimensional diagram with label $\xi$, as in Figure 1 can be used.

The symbols and notation of this paper are summarized in Table I.

![Fig. 1: 2D form of a time series.](image)

### Table I: Notations

| Notation | Description |
|----------|-------------|
| $\Xi$    | Time series of Malware |
| $0 \leq \xi \leq 1$ | Maliciousness level |
| $\phi_i: \tau_i$, $1 \leq i \leq n$ | availability at the time $\tau_i$ |
| $\xi$    | label |

D. Roadmap

The remainder of this paper is structured as follows. Section [II] provides a brief summary of related studies that have been designed to tackle malware analysis and visualization technique. Section [III] presents proposed architecture, include: clustering, probabilistic and deep learning approaches. A performance analysis of the proposed methods is presented in section [IV]. Finally, section [V] summarises the main achievements of the paper and gives some directions for future work.

II. Related Work

It has been proposed in the literature a number of ways for malware analysis that are based on machine learning techniques. We will highlight some of the research work that has been done using machine learning techniques to combat Windows malware. Malwares and other malicious software are a collection of instructions designed to do damage to businesses, their processes, networks, and infrastructure. Malware may be either an executable or a non-executable object, and its identification can be accomplished by either static, dynamic or hybrid examination of the infected system.

A. Malware analysis

Static analysis is the most basic and widely used method for determining the functionality of a system. In this method, the system monitors features of samples and tries to find malwares without running them. It is a technique for extracting features from an executable application, such as APIs, Permissions, hardware components and the Intent of the application. These extracted features may be used alone or in combination to detect malware. Machine learning techniques have been used to find malware in the case of static detection.

Dynamic approaches examine the functionality of an application in running time to determine if it is malicious or not. For example, the paper [2] came up with a way to detect Android malware called DroidCat that uses features from method calls and inter-component communication (ICC) to make a multi-class classifier and figure out if an app is malicious or not.

Hybrid approaches combines features of both static and dynamic analysis in a single procedure. Applications that become malware during runtime cannot be detected by static techniques, while malware that conceals its harmful behavior during operating cannot be detected by dynamic methods. Therefore, some authors develop their detection model using a combination of both dynamic and static techniques. They will be able to minimise the downsides while maximising the benefits of both in this manner.

Throughout literature, there are more researches have been done in this area. In addition to ransomware [3], Trojans [4], Keyloggers [5], Backdoors [6], Launchers [7], there...
are other categories of malware. Both signature-based and behavior-based techniques are used for malware detection. The signature-based approach is useful for identifying known threats without producing an excessive number of false alarms [8], but it needs regular manual updates of the database containing rules and signatures. According to [9], a malware detection strategy in the Internet of Things environment based on a similarity hashing algorithm has been presented. The scores of binaries were computed in the proposed approach in order to determine the similarity between malicious PEs.

Hybrid techniques combine aspects of both static and dynamic analysis. Static methods can’t find apps that become malware during runtime, and dynamic methods can’t find malware that hides its bad behaviour during operation. As a result, some papers use a combination of both dynamic and static approaches to design their detection model. This way, they can minimise the drawbacks and maximise the benefits of both.

B. Visualization technique

Image-based malware classification algorithms can overcome code obfuscation or encoding difficulties, comparable to and more advanced than static analysis approaches [8]. These image-based methods employ static analysis techniques to turn malware samples into image representations [10]. Conti et al. [11] conducted the first research on the visualisation of binary data as images. They demonstrated how to convert binary data into visual images, referred known as “byte-plots,” in order to increase the capabilities of text-based hex editors. Instead of depending on behaviour or signatures, researchers utilise transformation methods to turn raw malware binaries into pixel-based representations that may be used by machine learning algorithms to classify malware [12].

One developing method in machine learning for malware detection is to employ visual features from malware samples in conjunction with powerful machine learning algorithms in computer vision to maximise the benefits of advanced machine learning algorithms [13, 14]. The majority of these methods convert a binary file to an image by reading the binary file’s byte sequences as grayscale pixel values.

There was a study done by Yajamanam et al [15] that looked at the GIST-based byte-plot method with a limited set of features. Further testing was done with deep learning and Tensorflow. Results were similar to the original research by Nataraj et al. [10], but they used a smaller set of 60 features instead of the original 100 features. This is how Le et al. [17] used convolutional neural networks to look at binary files from the Microsoft Malware Classification Challenge dataset (Ronen et al., [18]) that had been turned into byte-plot images. Results from the data show that 98.8% of the validation data was correct with a processing time of about 20ms per sample.

The researchers in [19] used local gray-level co-occurrence matrices and global colour moments to extract features from grayscale and colour byte-plots. Classifiers such as Random Forest, Support Vector Machine, and K-Nearest Neighbor were then trained on these features. The studies in this paper used a dataset of 7087 malware samples from 15 various malware families. The Random Forest classifier’s accuracy, precision, recall, and f-measure were all more than 97%.

Unlike the work on byte-plots, there has been relatively little study on how to identify malware using space-filling curves. The authors of [20] devised a method for classification malware based on its nature, which makes use of Hilbert curves and a Self-Organizing Incremental Network. Experiments were conducted on 180 samples, with 78 of them proving to be completely innocuous. The malware was classified into these classes: Trojans, ransomware, and unknown. Even though the author claimed that the Hilbert curve was 89% correct, if you look at just 180 samples, it’s not apparent whether the Hilbert curve is an effective method of classifying malware. When working with smaller datasets, overfitting is more likely to occur, which implies the model will not be able to perform effectively with additional samples.

Vu et al. [21] devised a method that combined statistical and syntactic elements, which were then translated into an image format using a variety of layout functions, such as space-filling curves, to create a visual representation of the infection. An experimental dataset of 16,000 excellent and poor images was used to evaluate the technology, which was named hybrid image transformation (HIT). HIT was tested using four distinct layout functions, including two SFC implementations, Hilbert and Hcurve, which were both employed in the testing. According to the authors, their hybrid approach and Hilbert SFC layout function are the most accurate methods for achieving 93 percent accuracy. Throughout this essay, only two categories of objects are considered: those that are good and those that are harmful. It does not introduce malware onto a family’s network. When it came to categorization, the only way to determine how well you did was by your accuracy. It would have been possible to utilize other metrics to demonstrate how effectively the method functioned, such as accuracy, recall, and F1-score. It is possible that accuracy alone will not be sufficient to demonstrate how effectively it worked. Aside from that, there was no testing done with a holdout dataset to assess how well the model might generalize to new data.

III. PROPOSED METHOD

The method proposed in this paper is dedicated to monitoring target programs during $\tau$ seconds, which is called monitoring time. During this time, applications invoke extracted features by the monitoring system as a time series that is accessible using known datasets. The proposed method maps the samples into an image with dimensions $n_0 \times \tau \times \ell$ using algorithm [1] where $n_0$ and $\tau$ represent the number of extracted features and the monitoring time, respectively. After obtaining the images, different methods analyze them, and by introducing a more efficient method, the problems of the previous method are solved. Figure 2 shows this process.

A. Clustering Approach

In this section, images are analyzed and their labels are extracted using the clustering technique. To calculate labels, a clustering image that contains malware information is needed. This image is a $n_0 \times \tau \times \ell$ black and white image $CI$
Algorithm 1 Proposed malware detection algorithm.

Required matrices for $\ell$ maliciousness levels:
1: Set a zero $n_{\ell}$ by $\tau \times \ell$ matrix as $CL$, for clustering image.
2: Set a zero $n_{\ell}$ by $\tau \times \ell$ matrix as $PI$, for Probabilistic image.
3: Create a zero $1$ by $\ell$ matrix as $CW$, for the weight of clusters.
4: Create a zero $1$ by $\ell$ matrix as $CL$, for the label of clusters.
5: Create $\ell$ empty folders named $1, \cdots, \ell$, for storing deep learning images.
6: for $\Xi \in \text{Trainset}$ do
7: Set the label of $\Xi$ as $\xi$.
8: Set the Target Cluster $TC$ as: $TC = \{\ell \times \xi\} + 1$.
9: $CW_{1,TC} \leftarrow CW_{1,TC} + 1$
10: $CL_{1,TC} \leftarrow CL_{1,TC} + \xi$
11: Set a zero $n_{\ell}$ by $\tau$ matrix as $Z$.
12: for $(\phi, \tau) \in \Xi$ do
13: Time series to image conversion:
14: $Z_{\phi,\tau} = 1$
15: Clustering image computation:
16: $CI_{\phi,TC\tau} \leftarrow 1$
17: $Z_{\phi,\tau} = 1$
18: for $\Xi \in \text{Testset}$ do
19: Set a zero $n_{\ell}$ by $\tau$ matrix as $Z$.
20: for $(\phi, \tau) \in \Xi$ do
21: $Z_{\phi,\tau} = 1$
22: end for
23: end for
24: Cluster-based Approach (CA):
25: Compute $j$ that minimizes $\sum_{i=1}^{n_{\ell}} \sum_{c=1}^{N} |Z_{\phi,ci} - CI_{\phi,TC\tau}|$
26: Return $\sum_{i=1}^{n_{\ell}} \sum_{c=1}^{N} |Z_{\phi,ci} - CI_{\phi,TC\tau}|$ as the estimated maliciousness level $\xi$.
27: Probability-based Approach (PA):
28: Compute $j$ that minimizes $\sum_{i=1}^{n_{\ell}} \sum_{c=1}^{N} |Z_{\phi,ci} - CI_{\phi,TC\tau}| \times \frac{PI_{\phi,TC\tau}}{CL_{\phi,TC\tau}}$
29: Return $\sum_{i=1}^{n_{\ell}} \sum_{c=1}^{N} |Z_{\phi,ci} - CI_{\phi,TC\tau}| \times \frac{PI_{\phi,TC\tau}}{CL_{\phi,TC\tau}}$ as the estimated maliciousness level $\xi$.
30: Deep learning-based Approach (DA):
31: Find the returned image of Convolution Neural Network (CCN) from folders $1, \cdots, \ell$.
32: Return the label of returned image as the estimated maliciousness level $\xi$.
33: end for

consisting of a number of black or white pixels. This image can be considered as a binary matrix that contains information about $\ell$ clusters. If the element in position $(\phi, j)$ is equal to 1, it means that there is at least one malware with label $\xi$, as eq. 2 that called feature $\phi_{i}$ at time $\tau_{i} = j - \tau \times \lfloor j/\tau \rfloor$.

$$\frac{1}{\ell} \times \lfloor \frac{j}{\tau} \rfloor \leq \xi < \frac{1}{\ell} \times \left( \frac{j}{\tau} + 1 \right)$$  \hspace{1cm} (2)

Figure 3 shows a clustering image of the data set used as the train set for proposed clustering approach (CA), which contains more than 80,000 malware for eight maliciousness levels. In this figure, the white pixels represent 1, and the black pixels represent 0. To detect malware, it is enough to get a visual representation of it, comparison with the clusters in the clustering image, and report the label of the nearest cluster as a malware label. The proposed clustering approach is summarized in algorithm 1.

B. Probabilistic Approach

In the previous section, the clustering approach through clustering image was briefly introduced for malware detection. The clustering image provides suitable information for analysts. For example, at the bottom of Figure 3 there is a black rectangle, which means that none of the features in this rectangle are called in any of the malware, which can be a useful point in identifying malware behaviours. For this reason, this section proposes another method to improving the performance of clustering image, called the probabilistic image. This image is very similar to the clustering image and can be represented by a $n_{\ell}$-by-$\tau \times \ell$ matrix $PI$ with integer elements. The element $PI_{\phi,j}$ on the position $(\phi, j)$ indicates that there were $PI_{\phi,j}$ malware in the used database, labeled by $\xi$, which called $\phi_{j}$ at time $\tau_{i} = j - \tau \times \lfloor j/\tau \rfloor$. Algorithm 1 summarizes how to create $PI$, and calculate the malware labels. Similar to the previous method, $\xi$ satisfies eq. 2 and the method called Probabilistic Approach (PA).

C. Deep Learning Approach

Although clustering and probabilistic methods can provide valuable information to malware analysts, these methods do not examine or link the different situations. For example, consider a malware that uses feature $\phi_{i}$ after 1 second of monitoring, and this malware is available in the used database to train the malware detection system. Suppose again that the same malware is present in the test dataset and uses $\phi_{i}$ after 2 seconds. In this case, the previous two methods verify $CI_{\phi_{i}2}$ and $PI_{\phi_{i}2}$ to assess the maliciousness level, while there is no valuable information in such positions. In other words, previous methods lose their effectiveness against dynamic feature delays. To solve this problem, deep learning-based methods seem to be a good solution. For example, pooling techniques in Convolution Neural Networks (CNN) address adjacent pixels. Therefore, this method converts the samples to images with black and white pixels, stores them according to their maliciousness levels in folders with labels $1, \cdots, \ell$, and examines them by CNN. The details of three method are integrated in algorithm 1.

D. Space & Time complexity

The proposed method stores several matrices and images, the sum of their dimensions determines the space complexity. The matrices stored in algorithm 1 are $CI, PI, CW, CL$ and $Z$, whose dimensions are $n_{\theta} \times \tau \times \ell, n_{\theta} \times \tau \times \ell, \ell, \ell$, and $n_{\theta} \times \tau$, respectively. In addition, the algorithm stores $N_{TR}$ images in different folders, each of which has dimensions $n_{\theta} \times \tau$. Therefore, the space complexity of algorithm 1 is $2 \times \ell \times (n_{\theta} \times \tau + 1) + (N_{TR}+1) \times (n_{\theta} \times \tau)$.

Similarly, the time complexity of algorithm 1 can be calculated. The training part of the algorithm consists of two loops in lines 6 and 12. So the time complexity of this part is equal to $N_{TR} \times \sum_{Z}$, where $\sum_{Z}$ is the size of $Z$. At the worst case, $\sum_{Z}$ includes $n_{\theta} \times \tau$ elements. So, $\sum_{Z} \leq n_{\theta} \times \tau$. In the test section, there are two loops in lines 19 and 21, and the complexity of lines 24, 26, and 28 is assigned to each approach. In line 24, the complexity for calculating $i$ is $n_{\theta} \times \tau \times \ell$, and in line 26, the complexity of calculating $j$ is the same. Finally, the time complexity of line 28 depends on the complexity of the used deep learning method, which is assumed to be $T_{CNN}$. 

As a result, the total space complexity is as eq. 3 and eq. 4 represents the time complexity.

\[ O\left(\max\{\ell, N_{TR}\} \times n_\phi \times \tau\right) \]  

3

\[ O\left(N_{TE} \times (n_\phi \times \tau \times \ell + T_{CNN})\right) \]  

4

The numerical value for these parameters is \( \ell = 10, n_\phi = 482, \tau = 60, N_{TR} = 86284, N_{TE} = 21572, \) and \( N_{TE} \times T_{CNN} \approx 5318 \) seconds per epoch, where \( N_{TE} \times T_{CNN} \) is required time for training and testing the dataset using CNN.

IV. EXPERIMENTAL EVALUATION

A. Simulation setup

This section compares the proposed method with the state of the art ones by a comprehensive database. The results in this paper are obtained using the database introduced in [22], which contains more than 100K malware. The malware was obtained by analyzing millions of executable files by 52 antiviruses over a period of 4 years and extracting 486 important features. These features fall into the following four categories:

1) Application Programming Interface (API) with 353 features, which is dedicated to accessing the basic functions of the operating system.

2) Directory with 4 features, which is dedicated to global configuration of operating system.

3) File System with 11 features, which is dedicated to operating system’s data organization.

4) Miscellaneous with 118 features, to risk level indication of an executable file.

The used dataset is available on UCI Machine Learning Repository [23]. Figure 4 shows the distribution of the maliciousness level \( \xi \). As this figure shows, there are more than 6K malware with \( 0.74 < \xi \leq 0.76 \). Also, the Cumulative Distribution Function for \( \xi \) is depicted in [5] which shows that 100 – 27.67 = 72.33% of the analysed malware is very dangerous with \( \xi > 0.5 \). All simulations of this paper are performed in MATLAB R2017a on a laptop with 2.2GHz i7-3632QM CPU and 8GB RAM.

B. Comparison of solutions

In the previous section, three approaches to detect maliciousness level of malware were suggested. This section introduces two methods for comparison. In 2020, Taheri et al. [24] proposed a similarity-based method for detecting malware which is based on Hamming distance. They showed that the static features used in malware are so similar to each other that
by examining the similarity of the samples with pre-identified dataset samples, malware can be identified with an accuracy of over 99%. This method has a better performance compared to fixed size clustering, automatic clustering, neural networks, SVM, etc. For this reason, the method proposed in this paper will be compared with First Nearest Neighbour (FNN) that has the least complexity among the methods in [24]. This method is easy to implement, and to calculate the maliciousness level of a test sample, it is enough to calculate its distance from all the members in the trainset. In other words, $\ell_i = \xi_i$, where $1 \leq k \leq N_{TR}$ minimizes eq. (5). In this case, $\xi_i$ is the label of $k^{th}$ sample in the trainset with $N_{TR}$ elements. $Z'$ is the image form of the test sample, and $Z^k$ is the the same for $k^{th}$ sample.

$$\sum_{i=1}^{n} \sum_{j=1}^{\tau} |Z'_{i,j} - Z^k_{i,j}|$$

(5)

In 2021, Ritter and Urcid [25] published a very interesting study on lattice algebra, which deals with the applications of lattice algebra in image processing, pattern recognition, and artificial intelligence. As a second method to compare with the proposed method, Lattice-based Associated Memory (LAM) is used, which has a stronger mathematical background and much higher efficiency than other methods such as Hopfield Memory. This method integrates the trainset instances into an $n_{\Phi}$-by-$\tau$ image LAM as in eq. (6) and uses eq. (7) to estimate the maliciousness level $\xi'_i$.

$$LAM_{i,j} = \max\{\xi'_i - Z^k_{i,j} : 1 \leq k \leq N_{TR}\}$$

(6)

$$\xi'_i = \min\{Z'_{i,j} + LAM_{i,j} : 1 \leq i \leq n_{\Phi}, 1 \leq j \leq \tau\}$$

(7)

C. Test metrics

The most well-known evaluation criterion is mean cumulative absolute error (MCAE), which is shown in eq. (8) This criterion shows the average difference between the actual maliciousness level and the estimated one.

$$\frac{1}{n} \sum_{i=1}^{n} |\xi_i - \xi'_i|$$

(8)

In eq. (8), $n$ represents the size of test set, $\xi_i$ represents the actual maliciousness level of $i^{th}$ test, and $\xi'_i$ represents the estimated one.

The next section shows that the MCAE is not sufficient to demonstrate the performance of the proposed method, and it is easy to suggest methods that have a low MCAE but are not suitable for practical purposes and provide unreliable results. For this reason, MCAE and Confusion Matrix are used simultaneously in evaluating the proposed method. This matrix displays valuable information about the performance of the proposed method, the structure of which is shown in Figure 6. In $i^{th}$ row and $j^{th}$ column of confusion matrix there is $\lambda_{i,j}$ which means that in evaluation process there were $\lambda_{i,j}$ samples that had the true label $\ell_i$, while the proposed method estimates label $\ell'_j$. In evaluating the proposed method, the maliciousness levels are divided into 10 parts and if $(i - 1)/10 \leq \xi'_i < i/10$, $\ell_i = i$ is considered, where $1 \leq i \leq 10$. The same thing is for $\ell'_j$.

The performance evaluation criteria by the confusion matrix are based on the four numbers shown in Figure 7. To calculate these numbers for a target label, the confusion matrix is divided into four regions, represented by $R_1$, $R_2$, $R_3$, and $R_4$, which show True Positive, True Negative, False Positive, and False Negative, respectively. Using these values, different test metric can be obtained that show the performance of the proposed method well. Some of these metrics are summarized in Table II. In addition to the metrics introduced in this section, this section introduces two other metrics. A malware detection method is called conservative when eq. (9) is established ($R_6 \leq R_5$), otherwise it is called a loose detector. In this case, $R_5/R_6$ is called conservativeness ratio. Finally, the total
accuracy of an algorithm can be expressed by eq. 10.

\[
R_5 = \frac{\sum_{\ell=1}^{\ell} \sum_{r+1}^{\ell+1} \lambda_{r,c}}{\sum_{\ell=1}^{\ell} \sum_{c=1}^{c} \lambda_{r,c}} \geq 1
\]

\[
R_6 = \frac{\sum_{\ell=1}^{\ell} \sum_{r+1}^{\ell+1} \lambda_{r,c}}{\sum_{\ell=1}^{\ell} \sum_{c=1}^{c} \lambda_{r,c}} \geq 1
\]

In the next part, the methods introduced in this article are evaluated using the metrics and their pros and cons are examined.

D. Experimental results

In this section, the proposed methods are evaluated and the dataset is divided into two parts. The first one is for training and the second, which constitutes 20% of the data, is used to evaluate. The result of the first experiment is summarized in Figure 8 which deals with the MCAE of clustering-based approach. This figure shows that in the first 60 evaluation samples, the average error has an upward trend while it gradually decreases so that after evaluating 400 samples, the average error does not change much. The average error after evaluating 21572 samples is 41.59%, which is a large error.

Figure 9 shows the Confusion Matrix of the clustering-based approach. Figure 10 shows the MCAE of similarity based approach. Figure 11 shows another result that can be used to improve the MCAE of the proposed methods. As Figure 11 shows, the weights of columns 2 and 4 are zero, which means that no \(\xi'\) is estimated for similarity-based approach. Similarly, the weight of 6th column is 4, and the weight of 3rd, 5th, and 10th columns is one. These results show that the variety of malware with maliciousness level \(\xi < 0.1\) and \(0.6 \leq \xi < 0.9\) is so great that there is always a sample in them that bears the closest similarity to the samples being evaluated. The total accuracy for similarity-based approach is 15.79%, which is so better in comparison to clustering one.

Figures 12 and 13 show the results of the probabilistic approach evaluation. The general trend of MCAE in this approach is similar to the clustering method, while its Confusion Matrix behavior follows the similarity-based method. The average error of this method fluctuates per 1000 evaluated samples.
samples, later it does not change significantly, and eventually converges to 17.6%, which has less error compared to the previous two approaches. As Figure 13 shows, the Confusion Matrix of this approach has a large number of zero columns, which was also observed in the similarity-based one. In this figure, there are only three non-zero columns $C_3$, $C_9$, and $C_{11}$, where $C_8$ has more weight. This figure shows that with 97.67% probability, the probabilistic approach identifies all evaluated malware as a malware with maliciousness level $0.7 \leq \xi' < 0.8$. The total accuracy of this method is 25.51%, which has a much better performance compared to the previous two approaches.

The number of non-zero columns in the lattice-based Confusion Matrix reaches its peak so that there is only one non-zero column in this matrix, which means that the output of this method with is always a constant maliciousness level $c$. By analyzing eq. 6 and eq. 7, the constant output of this method can be calculated. In the used database, $Z_{i,j} \in [0, 1)$ and $0 \leq \xi < 1$. Assume that in eq. 7 $LAM_{i,j}$ is a fixed number $c$. In this case, $Z_{i,j} = 1$ and $LAM_{i,j} = c$, so $\xi' = \min(0 + c, 1 + c) = c$ means that the output of this method is always equal to $c$. In evaluating lattice-based approach, the method returns $c = 0.9828$. Now the question is: If a particular method, such as in lattice-based approach, always provides a fixed output $c$, is this output always the best possible output that keeps the error to minimum?

Figure 14 shows the MCAE for 10000 fixed maliciousness levels. As this figure shows, for $c = 0$, the lattice-based method is a loose approach that identifies all dangerous malware as harmless, the MCAE is over 60%, which is very high. As $c$ increases, the mean error decreases so that at $0.6613 \leq c \leq 0.6697$ the mean error reaches the lowest possible value of 17.1%. As shown in Figure 5 72.33% of the evaluated malware are high-risk, and to reduce the average error in the presence of a constant maliciousness level, the samples should be estimated at $c > 0.5$, as Figure 14 confirms.

The results of the last experiment are provided in Figures 15 and 16. As Figure 15 shows, in the first 100 tests, the average error has an upward trend, followed by a downward trend. The downtrend eventually converges to an average error of 11.1%, which is better than all previous methods. Confusion Matrix analysis in the deep learning method also indicates the high accuracy of this method compared to other methods. As Figure 16 shows, the weight of region $R_7$ is higher than other methods, which means that this method has the highest total accuracy of 45.1%. All methods in this section have been compared using different test metrics in Table III except for error rate which is equal to 1 - accuracy.

The bold values in Table III represent the best results for each class, which are assigned to the estimated maliciousness levels $\xi'$'s, and "-" represents the undefined test metric. For example, in the similarity-based method $R_1, R_3 = 0$, which makes the precision at $0.1 \leq \xi' < 0.2$ equal to $(0 + 0)/0$. As discussed earlier, the presence of zero column $C_i$ in the Confusion Matrix of an algorithm indicates that this algorithm is unable to generate output $\xi'$, and this appears "-" in the precision column corresponding to the $i$th estimated label.

In Table III, a per class metric score is assigned to each approach, which shows the number of best results obtained for each metric test in different classes. For example, the accuracy of the probabilistic approach with score 8 has the best value, except for $\xi' \in [0.5, 0.6) \cup [0.7, 0.8)$, which is in the best rank compared to all other approaches. This score corresponds to the number of bold values for each column. Note that the total accuracy of this method is 25.51%, which is at the second rank. The results show that except for the accuracy per class metric, deep learning is in the best position for all test metrics.
### TABLE III: Accuracy, Precision, Recall, F1-Score, MCAE, Conservativeness Ratio, and Total Accuracy values for evaluated dataset (Acc= accuracy; Pre= precision; Rec= recall; F1-S= F1-Score).

| Evaluated Approach | Clustering | Similarity-based | Probabilistic | Lattice-based | Deep Learning |
|--------------------|------------|------------------|---------------|--------------|--------------|
| Maliciousness level | Acc | Prec | Rec | F1-S | Acc | Prec | Rec | F1-S | Acc | Prec | Rec | F1-S | Acc | Prec | Rec | F1-S | Acc | Prec | Rec | F1-S | Acc | Prec | Rec | F1-S |
| 0.0 ≤ c < 0.1 | 0.518 | 0.0495 | 0.044 | 0.04 | 0.855 | 0.004 | 0.006 | 0.01 | 0.974 | - | 0 | 0 | 0 | 0.974 | - | 0 | 0 | 0.974 | - | 0 | 0 | 0.974 |
| 0.1 ≤ c < 0.2 | 0.865 | 0.0394 | 0.017 | 0.012 | 0.794 | - | 0 | 0 | 0.974 | - | 0 | 0 | 0 | 0 | 0.901 | 0.345 | 0.019 | 0.035 |
| 0.2 ≤ c < 0.3 | 0.871 | 0.0569 | 0.011 | 0.012 | 0.954 | - | 0 | 0 | 0.954 | - | 0 | 0 | 0 | 0 | 0.954 | 0.33 | 0.009 | 0.016 |
| 0.3 ≤ c < 0.4 | 0.828 | 0.0597 | 0.012 | 0.014 | 0.921 | - | 0 | 0 | 0.921 | - | 0 | 0 | 0 | 0 | 0.921 | 0.34 | 0.009 | 0.016 |
| 0.4 ≤ c < 0.5 | 0.81 | 0.0383 | 0.029 | 0.019 | 0.91 | - | 0 | 0 | 0.91 | - | 0 | 0 | 0 | 0 | 0.91 | 0.341 | 0.029 | 0.035 |
| 0.5 ≤ c < 0.6 | 0.839 | 0.1289 | 0.043 | 0.037 | 0.872 | - | 0 | 0 | 0.872 | - | 0 | 0 | 0 | 0 | 0.872 | 0.357 | 0.059 | 0.125 |
| 0.6 ≤ c < 0.7 | 0.824 | 0.2383 | 0.059 | 0.011 | 0.752 | 0.153 | 0.024 | 0.034 | 0.835 | - | 0 | 0 | 0 | 0 | 0.835 | 0.319 | 0.105 | 0.17 |
| 0.7 ≤ c < 0.8 | 0.782 | 0.2461 | 0.011 | 0.011 | 0.74 | 0.172 | 0.018 | 0.031 | 0.912 | 0.288 | 0.016 | 0 | 0 | 0.79 | 0.295 | 0.14 | 0.228 |
| 0.8 ≤ c < 0.9 | 0.841 | 0.3533 | 0.039 | 0.018 | 0.862 | 0.169 | 0.036 | 0.054 | 0.23 | 0.847 | 0.742 | 0.033 | 0.052 | 0.84 | 0.349 | 0.011 | 0.118 |
| 0.9 ≤ c ≤ 1.0 | 0.964 | 0.64 | 0.0 | 0.0 | 0.964 | 0.1 | 0.0 | 0.0 | 0.964 | 0.1 | 0.0 | 0.0 | 0.964 | 0.1 | 0.0 | 0.0 | 0.964 | 0.1 | 0.0 | 0.0 | 0.964 |
| Per class metric score | 1 | 0 | 1 | 5 | 2 | 1 | 1 | 1 | 8 | 1 | 1 | 1 | 7 | 0 | 1 | 1 | 4 | 7 | 6 | 6 |

| | MCAE | 41.59% | 26.36% | 17.6% | 36.93% | 3.012 | 3.874 | 26.36% | 3.012 |
| | Conservativeness Ratio | 0.203 | 3.5572 | 3.012 | 26.36% | 3.012 | 3.874 | 26.36% | 3.012 |
| | Total Accuracy | 7.31% | 15.79% | 25.51% | 3.63% | 46.1% | 46.1% | 46.1% | 46.1% |

### V. DISCUSSIONS

In the previous section, several methods for malware detection were analyzed, and the proposed method was compared with similar ones. This section discusses other ideas.

In the previous section, the proposed solution was compared with the similarity-based approach, presented in [24]. In this comparison, the samples were first converted into an image and then the method was applied to the obtained images. This approach is for detecting malware using static features, while the proposed method is based on dynamic ones. For this reason, instead of calculating the images using the proposed method, a vector corresponding to the extracted features can be used, and then the similarity-based method can be applied. In this case, the features call times are removed and the resulting vector contains the maliciousness level with the features shown in eq. [11]

\[ Xi = [\xi, \phi_1, \tau_1, \ldots, \phi_n, \tau_n] \rightarrow [\xi, \phi_1, \ldots, \phi_n] \]  

After this conversion, the similarity-based method can be applied to the calculated vector. Unfortunately, our implementation shows that this method suffers more errors than the image-based method, and therefore was not described in detail in the previous section.

The next point is about the clustering image shown in Figure 3. As mentioned earlier, this image gives us a lot of information about malware. For example, the black lines in this image indicate that there are features in the dataset that do not affect any of the clusters. Black columns, on the other hand, indicate that there are times when no feature is called. By removing these two black areas, the dimensions of the images can be reduced. Our implementation shows that this increases CNN 4 times faster. Unfortunately, this reduced the accuracy of the deep learning method by 4%, which is why it was not covered in detail in the previous section.

The next point is to analyze the results of the lattice-based method. As stated in the previous section, the outputs of this method are always \( c = 0.9828 \). The reader may be asked: what this number is and why labels are always estimated with this number? This number is the largest label in our dataset. In eq. 11, \( \xi \) represents this number. This number is the largest label in our dataset.

Now consider the second train instance \( Z^2 \) with label \( c \). Among the situations in which \( Z^2 \) contains 1, \( Z^2 \) may be zero. Similar to the previous case, \( Z^2_{i,j'} \) makes \( LAM_{i,j} \) to be zero. This continues until eq. [13] is established.

\[ \exists i \leq i \leq n_q, 1 \leq j \leq \tau, 1 \leq k \leq n_{max} \quad s.t. \quad Z^2_{i,j} = 0 \]  

Note that the binary matrices are sparse with a large number of zeros. For example, the size of \( Z^2 \) corresponding to the maliciousness level 0.9828 in our dataset is \( 482 \times 60 = 28920 \), while it has only 17 non-zero elements, meaning that at least \( 99.94\% \) of \( LAM \)'s elements are zero. Because \( c - 1 \) is a negative number, the remaining 17 elements will probably converge to the second largest number in the dataset, 0.9815. This analysis shows that at best, the evaluated samples are compared with a maximum of 18 maliciousness levels. Although this observation reduces the performance of the lattice-based method in detecting malware in this paper, it also has some interesting applications, which are discussed in [25]. Again, see the clustering image corresponding to the 8 clusters in Figure 3. In this image, there are pixels that are common in all clusters. These points will probably impose additional overhead on the system. Our focus is on pixels that are in only one cluster, which can lead to the emergence of interesting ideas. These points are referred to as kernels in lattice-based approach. Figure 17 shows the number of kernels in each cluster. In the kernel-based method, instead of using all
traintset samples, only the samples containing kernels can be used, which increases the efficiency. Implementing this method improves the accuracy of the lattice-based approach from 3.6% to 7.65% while still not being competitive with other compared methods. Also, the conservativeness ratio of lattice-based method was $\infty$, while in the kernel-based approach it is 0.0178, meaning that the kernel-based approach is a loose method that treats high-risk samples such as low-risk ones.

VI. CONCLUSION AND FUTURE DIRECTIONS

This paper introduced a method to detect online malware. The proposed method monitored the behavior of malware for a period of time, and extracts the dynamic called features during this period in the form of a vector. Due to the fact that we do not have much storage and processing space in IoT devices, proposed method in this paper transformed the extracted vector into a sparse binary image, and clustering, similarity-based, probabilistic, lattice-based, and deep learning approaches are introduced for the generated image dataset analysis. The results showed that the mean error of the clustering method is more than 40%, which puts this method in the worst rank, and on the other hand, deep learning with an error of 11.1% is in the best position. The evaluated methods were compared with 8 different metrics, among which the probabilistic method was in the best position in terms of accuracy per class metric, and deep learning with 45.1% accuracy in the remaining 7 metrics surpassed others.

This paper provides valuable results for future research. For example, we have shown that MCAE is not the only effective parameter in the performance of methods, and if regions $R_1$ and $R_2$ in the Confusion Matrix of a target class are equal to zero, there will be a zero column in this matrix that makes precision metric calculation impossible. This result can be used to suggest a multi objective optimization learning base method.

The results of this article can be used in other areas such as label flipping in adversarial machine learning. For example, we showed that the lattice-based approach always returns the maximum maliciousness level $c$ as output, for the used dataset. In this case, changing label $c' < c$ will not change the performance of this method. In addition, the conservativeness ratio can be used to identify vulnerable methods against adversarial targets. If this rate is too high, the probability of identifying dangerous samples increases, while if the rate is less than 1, this method returns smaller maliciousness levels as output, and this can encourage adversaries to produce malware with high malicious level.

REFERENCES

[1] S. K. J. Rizvi, W. Aslam, M. Shahzad, S. Saleem, and M. M. Fraz, “Proud-mal: static analysis-based progressive framework for deep unsupervised malware classification of windows portable executable.” Complex & Intelligent Systems, vol. 8, no. 1, pp. 673–685, 2022.

[2] H. Cai, N. Meng, B. Ryder, and D. Yao, “Droidcat: Effective android malware detection and categorization via app-level profiling,” IEEE Transactions on Information Forensics and Security, vol. 14, no. 6, pp. 1455–1470, 2018.

[3] D. W. Fernando and N. Komminos, “Fesa: Feature selection architecture for ransomware detection under concept drift,” Computers & Security, p. 102659, 2022.

[4] R. Fani and M. Saheb Zamani, “Runtime hardware trojan detection by reconfigurable monitoring circuits,” The Journal of Supercomputing, pp. 1–27, 2022.

[5] A. Singh, P. Choudhary et al., “Keylogger detection and prevention,” in Journal of Physics: Conference Series, vol. 2007, no. 1. IOP Publishing, 2021, p. 012005.

[6] J. Guo, A. Li, and C. Liu, “Backdoor detection in reinforcement learning,” arXiv preprint arXiv:2202.03609, 2022.

[7] G. X. Cun, S. Wang, D. Guo, S. Guan, B. Liu, and B. Wu, “Design and implementation of a high-speed real-time position detection system in electromagnetic coil launcher,” IEEE Transactions on Plasma Science, vol. 48, no. 9, pp. 3203–3210, 2020.

[8] I. Obaidat, M. Sridhar, K. M. Pham, and P. H. Phung, “Jadeite: A novel image-behavior-based approach for java malware detection using deep learning,” Computers & Security, vol. 113, p. 102547, 2022.

[9] A. P. Namanya, I. U. Awan, J. P. Disso, and M. Younus, “Similarity hash based scoring of portable executable files for efficient malware detection in iot,” Future Generation Computer Systems, vol. 110, pp. 824–832, 2020.

[10] M. Cho, J.-S. Kim, J. Shin, and I. Shin, “Mal2dd: 2d based deep learning model for malware detection using black and white binary image,” IEICE Transactions on Information and Systems, vol. 103, no. 4, pp. 896–900, 2020.

[11] G. Conti, E. Dean, M. Sind, and B. Sangster, “Visual reverse engineering of binary and data files,” in International Workshop on Visualization for Computer Security. Springer, 2008, pp. 1–17.

[12] Z. Cui, L. Du, P. Wang, X. Cai, and W. Zhang, “Malicious code detection based on cnns and multi-objective algorithm,” Journal of Parallel and Distributed Computing, vol. 129, pp. 50–58, 2019.

[13] F. Abdullayeva, “Malware detection in cloud computing using an image visualization technique,” in 2019 IEEE 13th International Conference on Application of Information and Communication Technologies (AICT). IEEE, 2019, pp. 1–5.

[14] R. U. Khan, X. Zhang, and R. Kumar, “Analysis of resnet and googlenet models for malware detection,” Journal of Computer Virology and Hacking Techniques, vol. 15, no. 1, pp. 29–37, 2019.

[15] S. Yajamanam, V. R. S. Selvin, F. Di Troia, and M. Stamp, “Deep learning versus gis descriptors for image-based malware classification.” in Icissp, 2018, pp. 553–561.

[16] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, “Malware images: visualization and automatic classification,” in Proceedings of the 8th international symposium on visualization for cyber security, 2011, pp. 1–7.

[17] Q. Le, O. Boydell, B. Mac Namee, and M. Scanlon, “Deep learning at the shallow end: Malware classification for non-domain experts,” Digital Investigation, vol. 26, pp. S118–S126, 2018.

[18] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, “Microsoft malware classification challenge,” arXiv preprint arXiv:1802.10135, 2018.

[19] J. Fu, J. Xue, Y. Wang, Z. Liu, and C. Shan, “Malware visualization for fine-grained classification,” IEEE Access, vol. 6, pp. 14 510–14 523, 2018.

[20] I. Baptista, “Binary visualisation for malware detection,” 2018.

[21] D.-L. Vu, T.-K. Nguyen, T. V. Nguyen, T. N. Nguyen, F. Massacci, and P. H. Phung, “Hitmal: Hybrid image transformation for malware classification,” Transactions on Emerging Telecommunications Technologies, vol. 31, no. 11, p. e3789, 2020.

[22] N. A. Huynh, W. K. Ng, and K. Ariyapala, “A new adaptive learning algorithm and its application to online malware detection,” in International Conference on Discovery Science. Springer, 2017, pp. 18–32.

[23] D. Dua and C. Graff, “UCI machine learning repository,” 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[24] R. Taheri, M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti, “Similarity-based android malware detection using hamming distance of static binary features,” Future Generation Computer Systems, vol. 105, pp. 230–247, 2020.

[25] G. X. Ritter and G. Uricd, “Introduction to lattice algebra: With applications in ai, pattern recognition, image analysis, and biomimetic neural networks.” 2021.