Empirical Analysis of Learning-based Malware Detection Methods using Image Visualization

Abdullah Sheneamer
Department of Computer Science
Jazan University
Jazan, Saudi Arabia

Essa Alhazmi
Department of Computer Science
Jazan University
Jazan, Saudi Arabia

James Henrydoss
Vision and Security Technology Lab
University of Colorado
Colorado Springs, USA

Abstract*—Malware, a short name for malicious software is an emerging cyber threat. Various researchers have proposed ways to build advanced malware detectors that can mitigate threat actors and enable effective cybersecurity decisions in the past. Recent research implements malware detectors based on visualized images of malware executable files. In this framework, a malware binary is converted into an image, and by extracting image features and applying machine learning methods, the malware is identified based on image similarity. In this research work, we implement the Image visualization-based malware detection method and conduct an empirical analysis of various learners for selecting a candidate learning classifier that can provide better prediction performance. We evaluate our framework using the following malware datasets, Search And RetrieVAl of Malware (SARVAM), Xue-dataset, and Canadian Institutes for Cyber Security (CIC) datasets. Our experiments include the following learning algorithms, Linear Regression, Random Forest, K-Nearest Neighbor (KNN), Classification and Decision Tree (CART), Support Vector Machine (SVM), Multi-Layer Perceptron (MLP), and deep learning-based Convolutional Neural Network (CNN). This image-visualization-based method proves to be effective in terms of prediction accuracy. Some conclusions emerge from our initial study and find that a Convolutional Neural Network (CNN) algorithm provides relatively better performance when used against SARvAM and various malware datasets. The CNN model achieved a high performance of F1-score and accuracy in the binary classification task reaching 95.70% and 99.50%, consecutively. The model in the multi-classification task achieved of 95.96% and 99.30% (F1-score and accuracy) for detecting malware types. We find that the KNN model outperforms other traditional classifiers.

Keywords*—*Malware detection; malware analysis; deep learning; machine learning; malware features

I. Introduction

Malware attacks are continuously evolving, and the spread of malware is ubiquitous and unstoppable. Attackers spread the malware through the Internet, email, and social media for the primary purpose of harming computers with a fraudulent intention. Typically, malware is an umbrella name for a set of malicious software, including viruses, worms, Trojan horses, spyware, etc. and are created to cause extensive damage to either data or systems or gain unauthorized access to a system or network. Cisco's recent cyber security threat report explores how cybercriminals exploited by building coordinated multi-step attacks using the following four types of attacks: crypto mining, phishing, Trojans, and Ransomware. These attacks received ten times more queries than any other type of attack. Building advanced software defense methods to mitigate the spread of malware is an absolute necessity. Malware detection is a process of detecting the presence of malware on a host system or identifying whether a particular program file is malicious or not malicious, i.e., benign [1]. The malware detectors play a crucial role in building early warning systems to thwart any attacks and prevent hackers from using computer systems during the zero-day attack period.

In the past, various research works built a plethora of malware detection methods that address techniques to implement advanced malware detectors. Nevertheless, malware authors continuously innovate new ways to penetrate the defense mechanism. A vast amount of research work exists that formulates the signature-based, behavioral-based, and machine learning-based malware detection approaches [2], [3], [4], [5], [6]. In a signature-based detection, each file is analyzed, assigned a signature or hash (a unique alphanumeric way to identify malware), and then added to the signature database, where it's used for comparison in subsequent malware incidents. This technique identifies specific patterns in the application to determine whether the file is malicious by verifying against a known set of signatures for matching patterns. These patterns can be syntactic, e.g., the sequence of instructions, or semantic-based, e.g., control or data flow properties. One of the critical aspects of any malware detection system is identifying whether a file is malicious or benign. The signature-based detection is a simple and widely used method built as AV (Antivirus) and malware detectors. A behavioral-based technique implements a dynamic environment where malware binaries are executed in a sandbox machine to extract the run-time characteristics to identify the malware. In a static or code analysis-based malware detection technique, the malware can be classified by using the code structure of the malicious code, e.g., use of control Flow Graphs (CFG) to identify the malware. The static method is the process of analyzing malware/binary without executing it. Its main objective is to extract useful information from the malware code structure, and it helps us get an idea of the type of malware and what the malware can do.

Signature-based approaches allow security analysts to identify the malicious component quickly, and so they are widely used by several commercial anti-virus (AV) companies. However, this technique's primary disadvantage is that they require a trained security analyst to manually write appropriate signatures that can be used to detect each malware family and load them in a signature database for run-time access. Unfortunately, this manual effort is error-prone and as well as time-consuming. Also, storing the whole world of signature-learning is a daunting task [7], [8], [9], [10]. Besides, when the malware mutates, the original stored signatures of malware become obsolete and unable to successfully detect the newly mutated malware due to the polymorphic and metamorphic methods that implement code obfuscation [11].

In contrast to signature-based detection, the machine learning-based (ML) technique aims to address this limitation of manual intervention and introduce automation by implementing a set of ML-based malware classifiers. Machine learning algorithms in the classification of malware rely on features extracted from binary files or a disassembled assembly code by using either the static code analysis techniques or dynamic analysis techniques where malware behavioral characteristics are studied during run-time and at the point of execution. This information is helpful for future analysis as it will allow us to analyze the sample efficiently. These techniques extract various features from the malicious samples and use standard machine learning algorithms to learn a classifier that labels the sample as either benign or malicious. However, there are still some challenges such as processing a large amount of malware, learning high-dimensional vectors, high storage usage, and low scalability in learning. Traditional approaches to malware detection using automatic classification are facing some limitations. The first one concerns feature extraction: static approaches are hindered by code obfuscation techniques, while dynamic methods are time-consuming, and evasion techniques often impede the correct execution of the code. The second limitation regards the building of the prediction models: the completeness of a training dataset may degrade over time as the malware authors evolve new techniques or can not be sufficient for some malware families or instances [12]. In addition, many malware runs independent of the operating system and executes its malicious code even before loads, running malicious.

The rest of the paper is organized as follows. Section II discusses related work. Our proposed framework are introduced in Section III. In Section IV, the framework's experiments are described. Discussion are covered in great details in Section V. Finally, the paper is concluded in Section VI.

II. Related Work

This section includes the relevant research associated with this project work comprising visualizing malware binaries and Machine-Learning (ML) based malware detection approaches. Malware detection has been widely practiced in the past decade. The detection approaches depend primarily on the following techniques: static - that relies on analyzing the code structure, call flow graphs of malware, the dynamic technique that addresses the behavioral characteristics of malware at run time execution in a sandbox environment during zero-day attack period [13]. Nataraj et al. [14] implemented the first framework on visualizing and classifying malware using image processing techniques. In their work, malware binaries are visualized as gray-scale images and observed that malware from the same families exhibited similarity in texture and layout. The authors conducted experiments on over 9,458 samples from 25 families of malware using image visualization and texture analysis. Nataraj et al. [15] also extended the approach to have malware binaries are represented as signals or images, and signal processing-based features are used to characterize malware. Han et al. [16] implemented visualization of malware images and performed a similarity calculation between images of malware variants. In this new classification method, they proposed a new classifier by first converting malware into gray-scale images and then applied a histogram similarity measurement to study the similarity of gray-scale image entropy maps. Han et al. [17] proposed a new malware family classification method that converts malware binary files into images and entropy graphs. Xue et al. [18] built a homology-based malware analysis using an ensemble of learning methods. Xue used gray-scale images, RGB color images, opcode sequences, and system flow graph-based image visualization methods and used Convolutional Neural Networks (CNNs) as base learners to perform bagging ensemble learning that extract features from malware images. Liu et al. [19] proposed an automatic malware classification and a new malware detection scheme using a clustering-based machine learning method. They implemented a new malware detection using Opcode n-gram based gray-scale images and feature extraction with Shared Nearest Neighbor (SNN)-based clustering algorithm on discovering new malware. Fu et al. [20] proposed visualizing malware using color images and extracted global texture and color features from the images for classification. A series of unique byte sequences are extracted from code and data sections of malware and using simhash functions converted to local features. K-Nearest Neighbors, SVM, and Random Forest methods are used to classify the malware. Makandar et al. [21] implemented malware classification methods that apply image processing techniques that use image textures-based features extraction from visualized malware binaries. Multi-resolution and wavelets are used to build effective texture feature vectors using Gabor Wavelet, GIST, and Discrete Wavelet Transform. They proposed using Support Vector Machine (SVM) based multi-class malware image classification. Singh et al. [22] implemented a CNN-based deep-network in building visualization-based malware detection methods. The use of ensemble learning techniques in malware detection is not new. Zhang et al. [23] implemented malicious code detection using multiple classifiers fusion and is not strictly dependent on specific malicious code. Menahem et al. [24] improved the prediction performance of malware detectors by combining the results of the individual classifiers into one final result to achieve overall higher detection accuracy. Recent research by [25] implemented an ensemble classification scheme of using both binary and multi-class classification as part of implementing intrusion detection solutions. There exists incomplete knowledge of class examples present during the training time. Scheirer et al. [26] introduced the open-set-based recognition method for a computer vision problem. The open-set classifiers are expected to detect all unknown classes present during testing but are not known to the model during training time. Open-set recognition describes the scenario in such a way that new classes (unknown unknown classes) unseen in training appear in testing, and requires the classifiers not only to accurately classify the *known classes*, but also to effectively deal with the *unknown* ones. In a malware survey paper, Rudd et al. [13] identified the following six flawed assumptions to use an open-set based malware detection instead of a closed-set one: intrusions are closed-set, anomalies imply class labels, static models are sufficient, no feature space transformation is required, model interpretation is optional, and class distributions are Gaussian. Henrydoss et al. [27]

implemented an open-set-recognition-based intrusion detection scheme.

Han et al. implemented a feature extraction method of malware binary using texture fingerprints index structure. This method achieves the highest accuracy of 85.77 %. Han et al., proposed a malware visualization method that uses both the static and dynamic code analysis technique with RGB color images. Using RGB image Opcode is generated from the malware by executing it, converting that opcode sequence into images for classification. The accuracy of this method yielded 98.96%. Wang et al. [28] addressed the problem of using small training sets. Opcode sequence extracted from the malware binary file was converted to an image and then normalized by histograms. Dilated and eroded PCA is applied to extract features for SVM-enabled classification. Tobiyama et al. [29], Kolosonjaji B et al. [30], Zhao et al.[23] implemented deep learning techniques CNN to classify malware images and achieve a prediction accuracy of 96%, 85.6%, and 96.7% respectively. These experiments were performed on a few malware samples, and these techniques are not evaluated against larger datasets. Liu et al., Azia Makandar et al., Huang et al. implemented malware classification based-on image analysis using multiple features, e.g., use of binary files, opcode sequences, and API call sequences, and resulted in an accuracy of 98.9%, 98.8%, and 99.51%. These methods used an advanced feature set ranging from 4000 to 50000 feature sets. Huang et al. [31], provided the state of the art classification performance evaluated against a large dataset that contains a training set of 4.5 million and a testing set of 2 million samples.

III. Framework

A. Malware Analysis using Image Visualization

Traditional malware detection methods are based on static and dynamic analysis techniques. The static analysis technique involves analyzing the disassembled code of a malware file. The dynamic code analysis and run-time-based methods use malware behavioral characteristics at the point of execution in a sandbox test environment. We adopt an image-visualization-based technique to implement malware detectors in this work. Basically, instead of detecting the malware binary as is where we convert them into an image and use computer vision techniques to recognize the image and detect the appropriate malware family. Visualizing malware is a new type of malware detection method. Using image processing is a simple but very effective method [14] of malware detection. Nataraj et al. [14] propose the initial framework of malware analysis using an image visualization-based malware detection technique. This method converts a malware binary into an image, using image processing techniques, and extract image features, e.g., texture and layout. Then a machine-learning-based image detector is used to identify the malware family that is solely based on malware images that exhibit similar behaviors. Fig. 1 depicts a detailed architecture and framework of the image-visualization based malware detection method. Firstly, malware binaries are converted or visualized as gray-scale images. They observe that for many malware families, images belonging to the same family aims to have similar image layout and texture. This is based on the assumption that most of the malware authors use an existing malware and modify the source code to create new malware. It has been observed that the Image visualization-based malware detection methods [19], [20], [18] yield a comparable detection performance in comparison to the regular static and dynamic malware detection technique [32], [33].

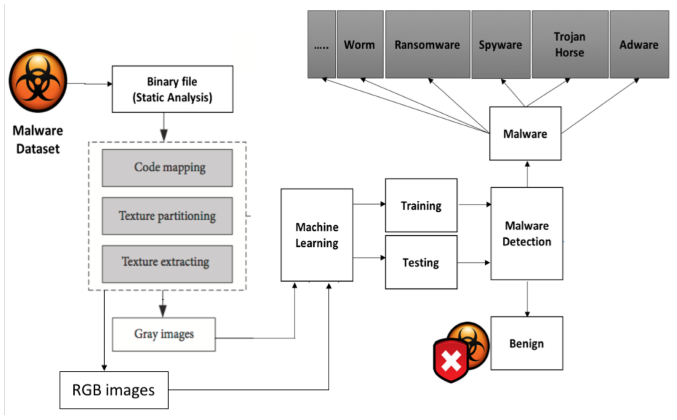


Fig. 1. Malware Detection Framework using Visualization Architecture.

The following covers the types of malware visualization methods: Gray-scale image visualization, RGB color image visualization-based methods that are widely used in the previous research works [19], [20], [18]. Also, using the malware binary file and other code analysis techniques like Opcode visualization and System Flow Graph (SFG) are also utilized [18]. As shown in Fig. 1 and Fig. 2, the system uses gray-scale and RGB color images of malware and computes the feature vector, i.e., fingerprint, to identify a malware binary. This fingerprint captures the structural, visual similarity between malware variants. The Opcode and SFG methods are out of scope for this work.

B. Gray-Scale Image Generation

In Fig. 2, a malware binary executable file is converted into a gray-scale image. A malware binary is converted to a vector of 8-bit unsigned integers and then organized into a two-dimensional (2D) array. This 2D data is visualized as a gray-scale image in the range [0,255] where 0 is black, and 255 denotes white. The image's width is fixed, and the height is allowed to vary according to the malware binary file size. Nataraj et al. recommend image widths for different file sizes based on empirical observations. We use their code base for converting and feature extraction for the Gray-scale image visualization prior to applying our proposed learning classifiers.

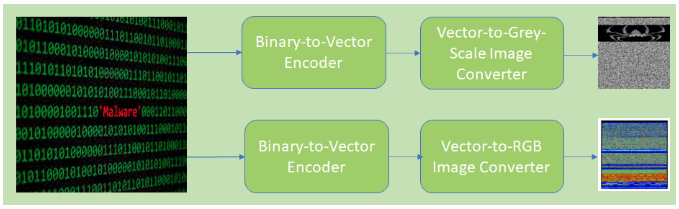

Binary-to-Vector Encoder
Vector-to-Grey-Scale Image Converter
Binary-to-Vector Encoder
Vector-to-RGB Image Converter
'Malware'

Fig. 2. Malware to Image Converter.

C. RGB-Color Image Generation

In Fig. 2, a malware binary executable file is converted into a RGB, color image. A malware binary is converted into a vector of 24 bits binary data (8-bit represents red color, 8-bit represents the green color, and 8-bit represents blue color) unsigned integers and then organized into a 2D array. This 2D data is visualized as a color image [0,255] where 0 is black, and 255 denotes white. The image's width is fixed, and the height varies according to the malware binary file size.

D. Malware Feature Extractor (MFE)

In this work, we conduct the malware detection approach using various machine learning algorithms on the features of the visualized image of the malware executable. We learn from the extensive research on signal and image processing techniques that implement compact signature extraction methods. The learning algorithms used in this work use image features extracted from these binaries. We consider techniques from the signal and image processing where compact structure extraction methods have been extensively studied. The image processing for content-based image retrieval has been extensively explored by Manjunath et al. [34], and scene classification by Olivia et al. [35] and Torralba et al. [36]. Manjunath et al.[34] propose image processing using texture information for browsing and retrieval of large image data. It uses the Gabor wavelet features for texture analysis and supports a comprehensive experimental evaluation. The Gabor features provide the best pattern retrieval accuracy [34]. We adopted a similar malware feature extraction method formulated using the GIST-based image features. The GIST method uses texture and spatial layout of an image [14], [37], [38]. Refer Fig.3 for the design details of the Malware Feature Extractor. Nataraj et al. founded the following feature extraction technique. Typically, a smaller resized or reshaped version of the image is used to compute the features [36]. Firstly, the binary executable file is converted to a discrete 1-dimensional signal by numerically coding every byte value as an 8-bit number that ranges from $0\breve{}255$. Then the signal is "reshaped" to a 2-dimensional gray-scale image with "d" being the width and "h" being the height of the reshaped image. During reshaping, the width "d'' and the height "h" are fixed depending on the number of bytes in the binary. The horizontally adjacent pixels in the image correspond to the adjacent bytes in the binary. The vertically adjacent pixels are associated with the bytes spaced by a multiple width of "d'' in the binary. Then the image is passed through various filters that capture both the short-range and long-range correlations in the image. The localized statistics are obtained by dividing the filtered images into non-overlapping sub-blocks from these filtered images and computing the average value on those blocks. This is called sub-block averaging. A compact signature is formed by concatenating the averages computed from all the filters. Typically, the features are extracted from the image's smaller "$resized''$ version.

The feature computation details are explained below. The image on which the feature needs to be extracted is defined using $(I(x, y))$. The GIST descriptor id computed by filtering the image through a bank of Gabor filters that are band-pass filters whose responses are Gaussian functions modulated with a complex sinusoid. The filter response $t(x, y)$ and the Fourier Transformed version $T(u, v)$ are defined by the

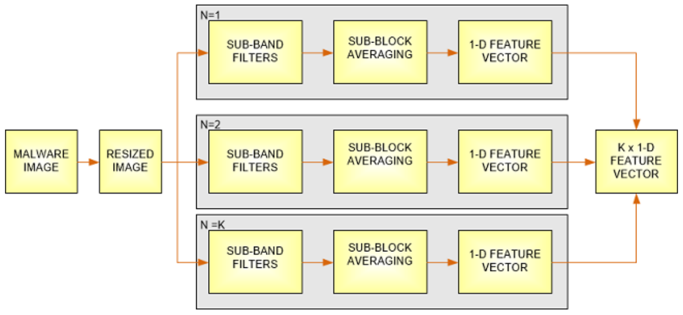


Fig. 3. Malware Feature Extractor.

following equations: Filter Response (FR) equation 1 and Fourier Transform of FR equation 2.

$$t(x, y) = \frac{1}{(2\pi\sigma_x\sigma_y)} exp[\frac{1}{2}(\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2})] \quad (1)$$

$$T(u, v) = exp[-\frac{1}{2}(\frac{(u-w)^2}{\sigma_u^2} + \frac{v^2}{\sigma_v^2})] \quad (2)$$

A filter bank, i.e., a block of filters, is created by rotating orientation and scaling the basic filter response function $t(x, y)$, resulting in a set of similar filters. Let S be the number of scales and O be the number of orientations per scale in a decomposed multi-resolution image. An image is filtered using "k'' such filters to derive "k'' filtered images. To conduct our experiment, we select the number of filters, $k = 20$ with a number of scales $S = 3$ and in which the first two scales have 8 orientations (O=8), and the last one has four orientations, i.e., $O = 4$. Each filtered image is further divided into $B.B$ sub-blocks, and the average value of a sub-block is computed and stored as a vector of length "L'' where $L = B^2$. Using this above method, "k'' vectors of length "L'' are created per image. These vectors are then concatenated to form a $kL-dim$ feature vector called GIST. We choose a B value of 4 to obtain a 320-dimensional feature vector in our work. When computing GIST descriptors, there is a key pre-processing step involved, that is, to reduce the image size to a square of dimensions SxS. In our work we choose a value of $S = 64$. An optimal value of S to be used in the computation because larger the value of S increased the computational complexity. Because of the sub-band averaging, this higher $S - vale$ does not significantly affect and strengthen the signature.

E. Machine Learning Approaches

We have explored a few learning classifiers to evaluate a better performing algorithm for implementing the malware detection function using virtualized images as part of this work. This section describes the theory and implementation of the machine learning algorithms that we used in this study: Naïve Bayes Classifier (NBC), Support Vector Machines (SVMs), Random Forests, k-Nearest Neighbours (KNN), CART, and MLP. As a note, all of the ML classifiers were trained on normalized data.

F. Evaluation Methods

The evaluation of machine learning classifiers is critical when studying the learning models and their performance. To evaluate the performance of the classifier models, we have used similar evaluation measures that are adopted in most of the previous research experiments that involve learning-based malware detection using visualized images of malware binaries. It covers the prediction accuracy and F1-score under varying conditions of input parameters. Most of the time, we use classification accuracy to measure the performance of machine learning models, and we have also used confusion matrices to compare the prediction accuracy and failures.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{3}$$

$$Precision(P) = \frac{TP}{TP + FP} \tag{4}$$

$$Recall(R) = \frac{TP}{TP + FN} \tag{5}$$

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{6}$$

In the above equations, TP, TN, FN, FN, and FP are true positives, true negatives, false positives, and false negatives. We use F1-score as the primary performance indicator to evaluate all the classifier models used in our experiments. F1 score is a single metric that combines both precision and recall. The precision or True Positive Rate (TPR) is a way to look at the accuracy of positive predictions performed by a classifier and can be defined as follows: precision = $TP/(TP + FP)$ where the True Positives (TP) is the number of true positives, i.e., correct prediction of a positive sample, and the False Positives (FP), i.e., the wrong prediction. But precision is used with another parameter called recall. Recall is defined by $TP/(TP + FN)$.

We have also used a confusion matrix table to study the performance of classifiers. The confusion matrix is a table with rows and columns that report false positives, false negatives, true positives, and true negatives. This allows a more detailed analysis than the mere proportion of correct classifications, i.e., prediction accuracy.

IV. Experiments

Our experiments include the following three step approach. Firstly, we convert the malware binaries into images. Secondly, we perform feature extraction using the GIST-enabled MFE. Finally, we perform the malware classification using the following machine learning methods: Linear Regression, Random Forest, KNN, CART, SVM, and MLP, CNN. We evaluate on the following three malware datasets; SARVAM [14] (25 malware families), Xue et al. [18] (10 malware families), and CIC [39] (six malware categories). These experiments leverage gray-scale image features and RGB Color image features to build models of different classifiers and evaluate their performances.

A. Datasets Summary

Fig. 4 shows malware families and categories distributions across all datasets. SARVAM dataset contains 9,339 instances broken into 25 malware families. The families distribution of malware are unbalanced with majority instances belonging to a family called *Allaple.A* as seen Fig. 4(a). In Xue et al. dataset, there are 10 malware families distributed in 5,314 instances. Fig. 4(b) presents malware family distribution with almost 20% of *Backdoor.Win32.Bifrose* as the top family. CIC is the smallest dataset which contains 439 instances and broken into six categories with a lot of families. This data shows that *Scareware* category includes the most instances among the families and *Botnet* includes the least instances as noticed in Fig. 4(c). Last, the benign dataset includes 1,024 instances collected from different sources and combined as one dataset. Finally, our experiments for the gray-scale images include all datasets while the RGB color images were just included for Xue and CIC datasets. We have not included the RGB color images as part of experiments and used only the gray-scale images due to unavailability of the original executable files of SARVAM dataset.

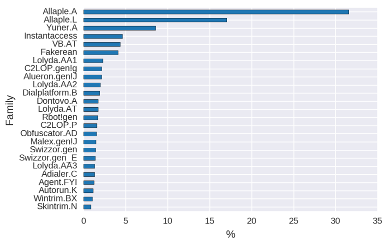
B. Models Setup

Our experiments are conducted in two phases. The first phase focuses on identifying malware from benign as binary classification tasks. The second phase involves a multi-class classification that identifies the individual malware family. We evaluate seven classification algorithms and compare between them based on four performance metrics stated above. Due to the limitation of datasets sizes, we used 10-fold cross-validation mechanism to train and test the classification algorithm. The second phase aims to detect the family or the category of malware as a multi-classification task using the most effective classification algorithm in the previous phase. This task is evaluated by confusion matrix in order to see how the actual unbalanced malware families and categories are truly and falsely predicted. Both phases were consecutively built to compare the results of gray-scale images and RGB color images.

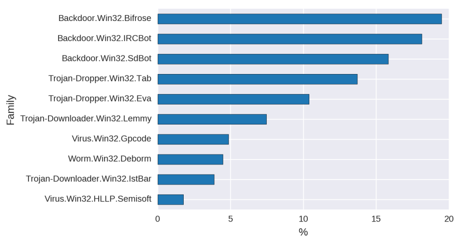
C. Binary Classification Performance

In this phase. we compare the performance results of the seven classifiers as described in Fig. 5 and Table I. This task aims to predict malware instances from benign instances. Each instance includes 320-dimensional feature vector with the class types. The total elements of the feature vector are broken into 960 (320x3) elements. We firstly measure the accuracy of each fold (using 10-fold cross-validation) in each classifier. We depicted the accuracy results for the folds through their quarterlies in the box plots as seen in Fig. 5. We found that K-Nearest Neighbors classifier outperform others in the accuracy results across all datasets, while support vector machines and Naïve Bayes classifiers perform closely low in most cases.

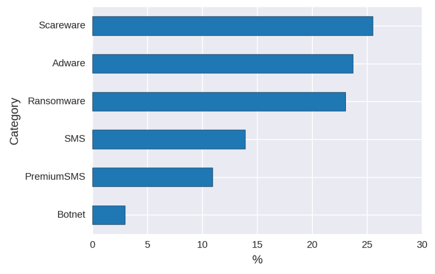
In a more nuanced view, the performance of binary classification for malware and benign can be described through four following metrics: average accuracy, precision, recall and F1-score as seen in Table I. Whilst recall denotes the ability to locate every relevant instance in classification datasets, precision denotes the number of data points a classifier classifies



(a) (a) Sarvam dataset

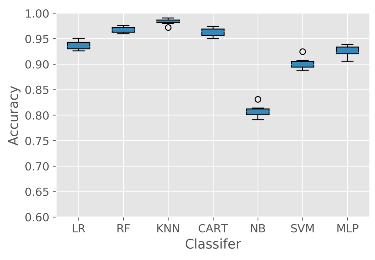


(b) (b) Xue dataset

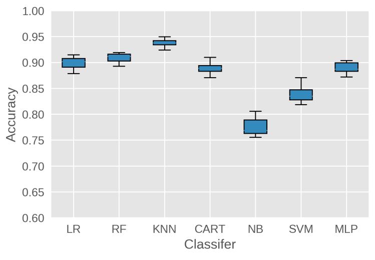


(c) (c) CIC dataset

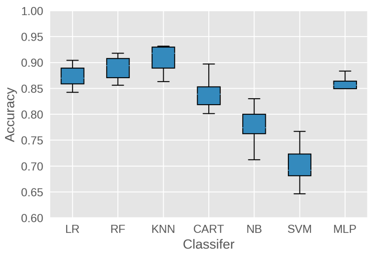
Fig. 4. Malware Families and Categories Distribution over All Datasets.



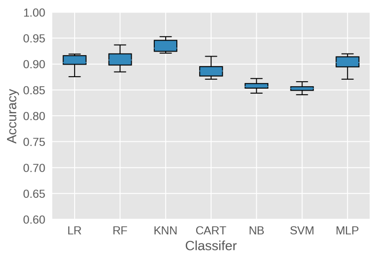
(a) (a) gray-scale: Sarvam



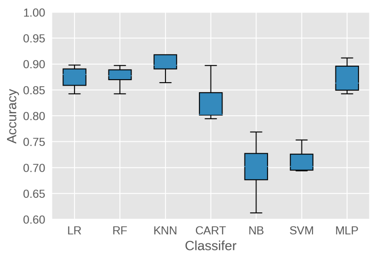
(b) (b) gray-scale: Xue



(c) (c) gray-scale: CIC



(d) (d) RGB Color: Xue



(e) (e) RGB Color: CIC

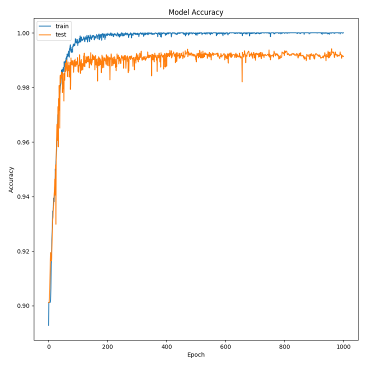
Fig. 5. Comparison of Seven Classifiers 10-fold CV Accuracy Results for Predicting Malware and Benign.

TABLE I. BINARY CLASSIFICATION TO PREDICT MALWARE AND BENIGN USING SEVEN CLASSIFIERS OF THREE DATASETS.

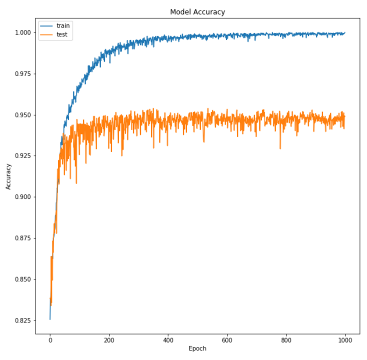
Sarvam						Xue								CIC							
Classifiers	Class	Avg.Acc.	Precision	Recall	F1-score	Avg.Acc.		Precision		Recall		F1-score		Avg.Acc.		Precision		Recall		F1-score	
						Gray-Scale	RGB Color	Gray-Scale	RGB Color	Gray-Scale	RGB Color	Gray-Scale	RGB Color	Gray-Scale	RGB Color	Gray-Scale	RGB Color	Gray-Scale	RGB Color	Gray-Scale	RGB Color
LR	Benign	93.69%	87.30%	42.29%	56.97%	89.92%	90.50%	82.33%	81.98%	48.24%	52.44%	60.84%	63.97%	87.36%	87.83%	91.18%	91.71%	90.82%	90.72%	91.00%	91.21%
	Malware		94.01%	99.33%	96.59%			90.76%	91.43%	98.01%	97.78%	94.25%	94.50%			78.78%	78.89%	79.50%	80.87%	79.14%	79.87%
RF	Benign	96.95%	95.64%	72.75%	82.64%	91.02%	90.68%	85.04%	84.15%	52.73%	53.42%	65.10%	65.35%	89.27%	86.60%	93.66%	90.60%	90.82%	91.31%	92.22%	90.95%
	Malware		97.09%	99.64%	98.35%			91.51%	91.61%	98.21%	98.06%	94.74%	94.73%			80.00%	79.35%	85.65%	77.90%	82.73%	78.62%
KNN	Benign	98.27%	98.84%	83.50%	90.52%	93.83%	93.59%	**88.56%**	86.58%	71.09%	71.19%	78.87%	78.14%	91.05%	90.29%	**96.47%**	95.66%	90.63%	90.33%	93.45%	92.92%
	Malware		98.22%	99.89%	99.05%			94.63%	94.63%	98.23%	97.87%	96.40%	96.23%			80.84%	80.04%	92.26%	90.43%	86.17%	84.92%
CART	Benign	96.40%	84.17%	80.47%	82.28%	88.91%	88.10%	64.23%	63.65%	63.48%	61.91%	63.85%	62.77%	84.89%	83.73%	90.09%	87.81%	88.77%	87.21%	89.42%	87.51%
	Malware		97.87%	98.34%	98.10%			92.98%	92.70%	93.19%	93.19%	93.08%	92.94%			74.67%	70.63%	77.22%	71.75%	75.92%	71.19%
NB	Benign	80.66%	26.00%	51.86%	34.64%	77.20%	86.01%	35.25%	60.03%	48.44%	39.45%	40.81%	47.61%	77.51%	69.79%	96.77%	94.21%	70.12%	60.35%	81.31%	73.57%
	Malware		94.08%	83.82%	88.65%			89.29%	89.06%	82.86%	94.94%	85.95%	91.90%			57.56%	49.69%	**94.53%**	91.34%	71.55%	64.37%
SVM	Benign	90.12%	0.00%	0.00%	0.00%	83.84%	85.33%	0.00%	**94.34%**	0.00%	9.77%	0.00%	17.70%	70.00%	69.99%	69.99%	69.99%	**100.00%**	**100.00%**	82.35%	82.35%
	Malware		90.12%	**100.00%**	94.80%			83.84%	85.17%	**100.00%**	**99.89%**	91.21%	91.95%			0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
MLP	Benign	92.59%	86.43%	28.61%	42.99%	88.85%	89.93%	79.45%	82.46%	39.26%	49.12%	52.55%	61.57%	86.33%	85.58%	88.98%	91.12%	90.72%	90.14%	89.85%	90.62%
	Malware		92.71%	99.51%	95.99%			89.33%	90.90%	98.04%	97.99%	93.49%	94.31%			77.33%	77.56%	73.80%	79.50%	75.52%	78.52%
CNN	**Benign**	**99.50%**	98.20%	**96.70%**	**97.50%**	**94.83%**	**95.46%**	85.28%	83.84%	**82.25%**	**87.03%**	**83.73%**	**85.40%**	**91.17%**	**93.75%**	95.94%	**99.31%**	**90.83%**	91.14%	**93.31%**	**95.05%**
	Malware		**99.60%**	**99.80%**	**99.70%**			**96.60%**	**97.65%**	97.26%	96.98%	**96.93%**	**97.31%**			**82.58%**	**85.26%**	**91.88%**	**98.78%**	**86.98%**	**91.53%**

as relevant where it really is relevant. There is a compromise in these two evaluation metrics to maximize, when increasing the recall, the model decreases the precision. In case we want to find an optimum balance of recall and precision, we use the F1-score to combine measurements for both metrics. We applied these performance metrics per class to evaluate the instance type (belonging to either malware and benign).

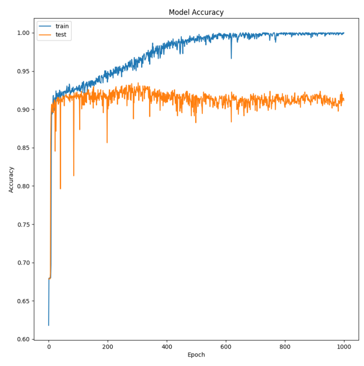
The K-Nearest Neighbor classifier (KNN) achieved in the average accuracy about 98.27%, 93.83%, and 91.05% for the gray-scale images of SARVAM, Xue, CIC datasets respectively, irrespective of the instance types, i.e., malware or bengin. It also achieved similar performance using the RGB color images of the two datasets, i.e., Xue and CIC. While the average accuracy metric cannot evaluate the performance of benign and malware separately, the performance of KNN vary across datasets. For the Xue dataset, F1-score on both gray-scale and RGB color images achieves closely better results in detecting malware (96.48% and 96.23%) than other classifiers except CNN classifier. While KNN F1-score for both gray-



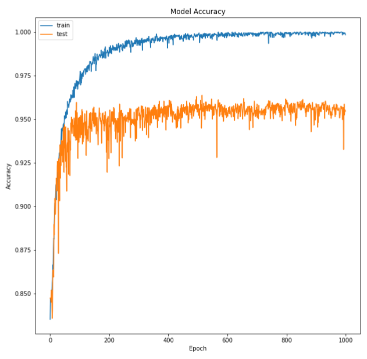
(a) Gray-scale: Sarvam dataset



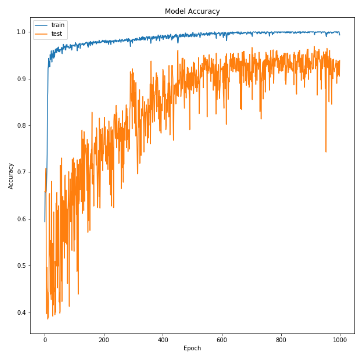
(b) Gray-scale: Xue dataset



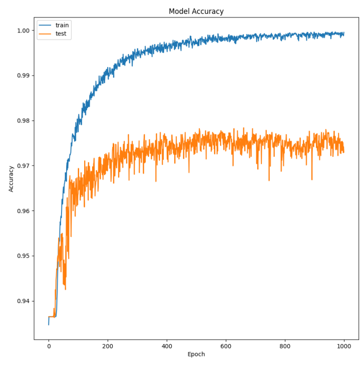
(c) Gray-scale: CIC dataset



(d) RGB Color: Xue dataset



(e) RGB Color: CIC dataset



(f) Gray-scale: All datasets

Fig. 6. Binary-classification CNN Accuracy Models based on Epochs.

scale and RGB color images in detecting benign achieves better results than others (78.87% and 78.14%), it preforms low compared to SARVAM dataset. Conversely, KNN model in CIC dataset can be accurately detecting more benign instances than malware instances for both gray-scale and RGB color images as clearly seen in the F1-score results for both benign (93,45% and 92.92%) and malware (86.17% and 84.92%).

While the tree-based classifiers especially Random Forest (RF) perform well in this classification domain in SARVAM dataset, Decision Tree (CART) results underperform the logistic regression (LR) in the remaining datasets. The performance of these classifiers across datasets follows the same pattern of KNN when comparing F1-score results for detecting malware and benign. In other words, the classifiers' performance for detecting more instances of malware than benign were seen in SARVAM and Xue datasets. In contrast, the results in CIC dataset show that classifiers can detect more benign instances than malware.

The Multilayer Perceptron (MLP) classifier performance results are seen in the median position and they can be improved over all datasets. They are close to the performance results in the logistic regression classifier.

Support Vector Machine classifier (SVM) usually fails to detect the minority class. This means it can learn only one class that reflects the majority. For example, benign instances are considered of the minority class in SARVAM and Xue datasets, and the F1-score results are either zero or below 10, and vice versa. In addition, Naïve Bayes classifier achieves the lowest results in all metrics over all datasets.

The Convolutional Neural Network classifier (CNN) achieved in the average accuracy about 99.50%, 94.83%, and 91.17% for the gray-scale images of SARVAM, Xue, CIC datasets respectively, irrespective of the instance types, i.e., malware or bengin. It also achieves similar performances using the RGB color images of the two datasets, i.e., Xue and CIC. Specifically, CNN F1-score (where there is a balance of precision and recall) in SARVAM dataset achieves higher results for malware instances than benign instances because of their falling off in their recall scores. This means that some instances of benign are predicted to be malware. In contrast, the high precision scores reflect that classifier correctly predicts the majority instances of each type, whether benign or malware. For the Xue dataset, F1-score on both gray-scale and RGB color images achieves closely better results in detecting malware (96.93% and 97.31%) than all other classifiers and F1-score for both gray-scale and RGB color images in detecting benign achieves better results than others (83.73% and 85.40%). In contrast, CNN model of CIC dataset can be accurately detecting more benign instances than malware instances for both gray-scale and RGB color images as clearly seen in the F1-score results for both benign (93.31%

and 95.05%) and malware (86.98% and 91.53%). Finally, the results show that the CNN classifier is the best algorithm to be used in the second phase of classification task. Binary-classification CNN accuracy models based on 1000 Epochs are shown in Fig. 6(a), (b), (c), (d), (e), and (f).

D. Multi-classification Performance

After evaluating binary classification tasks in the first phase, we selected the best classification algorithm that performs well in all metrics among all datasets to be used in this phase. Thus, we aim in this phase to measure the performance of the KNN classification task on identifying the individual malware family or category. Due to the variation in naming malware families on each datasets, we separately evaluated the model performance for each dataset. We chose the confusion matrix as a performance metric in order to see how the actual unbalanced malware families and categories are truly and falsely predicted. It also depicts where the model is confusing classes and mislabeling one as another.

The performance results of the KNN models for malware families are reflected in a confusion matrix as shown in Fig. 7, where the ordinate and abscissa are the number of the malware family or category. The abscissa indicates the actual malware family and the ordinate indicates the predicted malware family. The color gray patches in the figure indicate the similarity between the predicted instances and the actual instances under specific family or category. According to the ribbon on the right, the more dark the color is to the top, the higher the similarity is, and less dark it is to the bottom, the lower the similarity is.

In SARVAM dataset (available only for gray-scale images), the KNN model achieves a large probability of true positives over the majority of malware families depicted in the dark color of the right diagonal in Fig. 7(a). In the other hand, there are few families misclassified. For example, the figure shows that lot of instances of "*Swissor.gen*" malware are falsely predicted as "*Swissor.gen E*" malware (42%) and vice versa (34%). It seems the features vector of both malware families showing same patterns and have identical sequences of codes. Similarly, many "*C2LOP.P*" malware instances are misclassified by the model as "*C2LOP.gen!g*". This indicates that the model is biased toward the malware family that contains more instances and have identical sequences of codes with the other family. In other words, when a false prediction occurs, the predicted family is likely to belong to the same family series.

While the Xue dataset for the gray-scale images and the RGB color images are evaluated separately, the performance results of true positives and false positives are compatibly approximate among malware families as seen in Fig. 7(b) and Fig. 7(d). The KNN model for detecting malware family for this dataset achieves lower performance in the confusion matrix than for the SARVAM dataset. In addition, non malware families exceed 89% of the true positives of the total instances. For the CIC dataset, the KNN model also does not achieve a good results (Fig. 7(c) and Fig. 7(e)) compared to the models built for the other datasets.

We also train and test a model for multi-classification using CNN. For the SARVAM dataset, accuracy on gray-scale images achieves closely better results in detecting malware 99.30% than all other classifiers as shown in Fig. 8(a) and Fig. 9(a). For the Xue dataset, accuracy on both gray-scale and RGB color images achieves better results in detecting malware (87.79% and 88.09%) than all other classifiers as shown in Fig. 8(b) and (d) and Fig. 9(b) and (d) . For the CIC dataset, accuracy on both gray-scale and RGB color images achieves better results in detecting malware (78.51% and 85.93%) than all other classifiers as shown in Fig. 8(c) and (e) and Fig. 9(c) and (e)

TABLE II. MODEL GENERALIZATION: TESTING THE MODEL ON UNSEEN DATA TO PREDICT MALWARE AND BENIGN USING KNN AND CNN

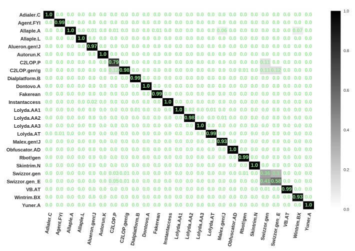
Accuracy	Classes	Precision	Recall	F1-Score
97.10% (KNN)	Benign	89.66%	61.54%	72.98%
	Malware	97.44%	99.52%	98.47%
97.56% (CNN)	Benign	82.4%	77.81%	80.18%
	Malware	98.5%	98.9%	98.7%

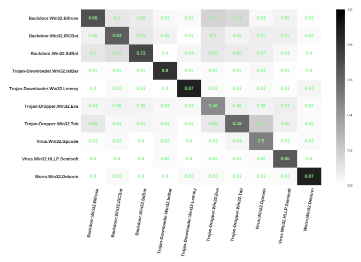
E. Deep Learning Experiments using CNN

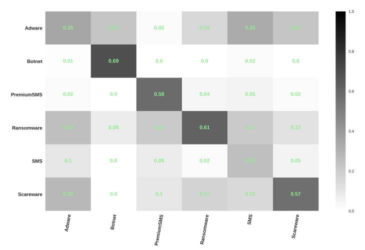
We also conducted deep learning experiments on visualized malware images using the Convolutional Neural Networks (CNN). After we generate grayscale and RGB images from Binary malware or benign executable files. we construct the CNN which has 24 layers (excluding the input layer), including 8 convolutional layers, 5 pooling layers, 6 dropout layers, 3 full-connection layers, and an output layer. All the convolutional layers use a 3×3 convolution kernel with a step size of 1; the number of convolution kernels in the eight layers are 8, 16, 32, 32, 64, 64, 128, and 256. Because the size of the feature map does not change when the feature map passes through a convolutional layer, a 1-pixel edge fill is performed on each input feature map in the convolution layer. We use all max pooling layers with a 2×2 sliding window and a step size of 2 as shown in Fig. 10. Because the last fully-connected layer of the CNN requires that the input feature maps should be the same size, the general CNN network structure needs to preprocess the image to unify the image size. We use a dropout regularization layer with 0.25 and 0.5 after each pair of convolutional and pooling layers to prevent CNN network overfitting. We also use ReLU and softmax activation functions for multi malware classification and sigmoid for binary malware classification.

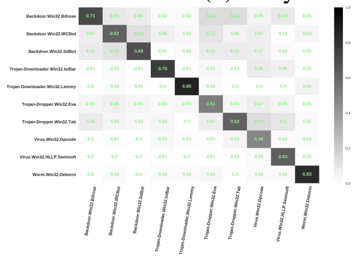
F. Multi-class Open-Set Recognition Performance

Most of the malware detectors used today fall under the category of closed-set assumption. In a closed-set operation, the data for training and testing are drawn from the same label space and from the same distribution. A large database containing malware signatures are used and signature vectors received are compared against the database leading to a binary classification scheme of malicious or benign files. If there is a new malware, i.e., the training and testing distributions are different, all these methods from closed-set detection will fail. One possible approach that can be extended from our empirical study is to use the Open-set recognition based approach to implement malware detection technique using visualized malware images.


(a) Gray-scale: Sarvam dataset


(b) Gray-scale: Xue dataset


(c) Gray-scale: CIC dataset


(d) RGB Color: Xue dataset

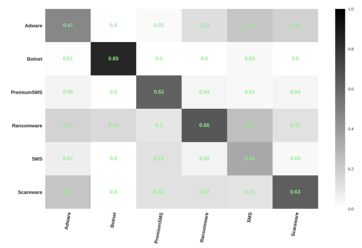
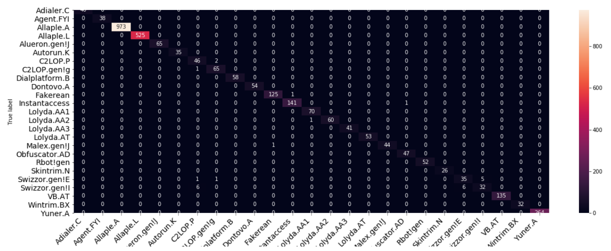
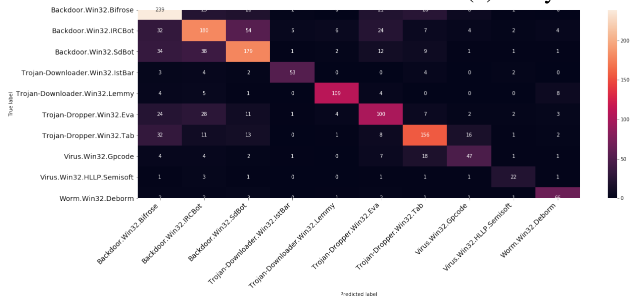
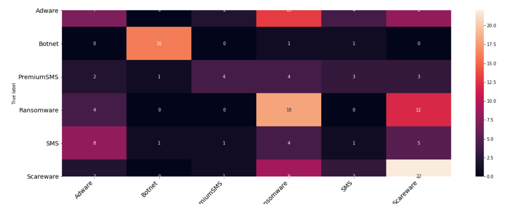
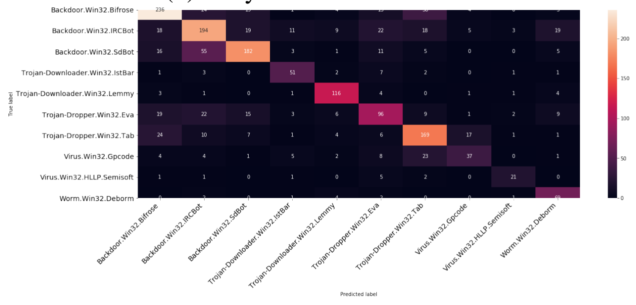

(e) RGB Color: CIC dataset

Fig. 7. Multi-classification Confusion Matrix of the KNN Classifier to Detect Malware Family and Category over Three Datasets.


(a) Gray-scale: Sarvam dataset


(b) Gray-scale: Xue dataset


(c) Gray-scale: CIC dataset


(d) RGB Color: Xue dataset

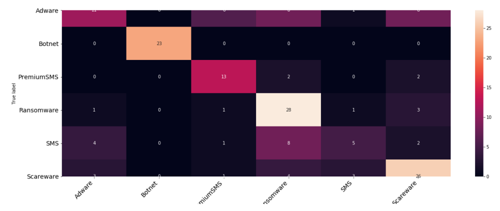
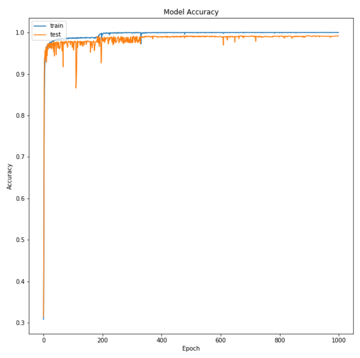
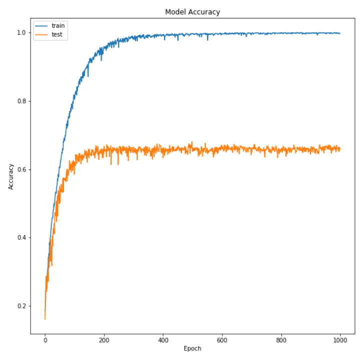

(e) RGB Color: CIC dataset

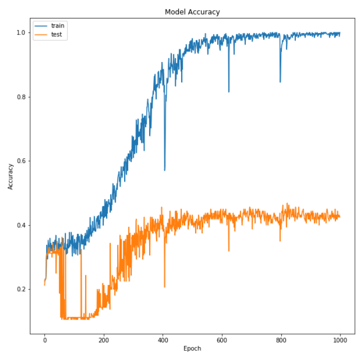
Fig. 8. Multi-classification Confusion Matrix of the CNN Classifier to Detect Malware Family and Category over Three Datasets.



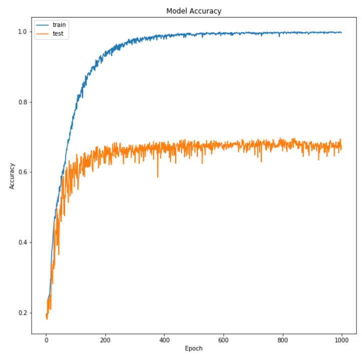
(a) Gray-scale: Sarvam dataset



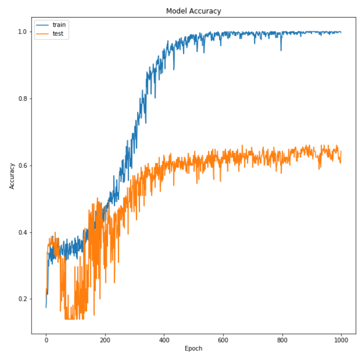
(b) Gray-scale: Xue dataset



(c) Gray-scale: CIC dataset



(d) RGB Color: Xue dataset



(e) RGB Color: CIC dataset

Fig. 9. Multi-classification CNN Accuracy Models based on Epochs.

G. Model Generalization

Unlike other works, we built a generalized KNN and CNN models using 66.67% of the three datasets for the binary classification task. Then, we tested the model on the remaining combined datasets (33.33%) as unseen data. Our splitting mechanism applies the stratified sampling method to avoid the bias toward a majority class. This approach is more solid for evaluating the performance of the model. Moreover, combining datasets from independent sources will examine whether the model can be generalized for any cases. Table II shows the performance of the models tested on unseen instances of benign (338) and malware (4,981). In general, the models achieves above 97% of the accuracy where it is better detecting malware than benign. To investigate the model in depth, let us assume that malware cases are positive and benign cases are negative. The models were tested on 4,981 malware cases and 338 benign cases as unseen cases. The model of KNN predicted correctly more than 4,000 malware instances as true positive cases while there are 24 malware instances predicted as benign. In the other side, the model of KNN predicted truly 208 benign instances while the remaining are predicted as malware. The model of CNN predicted correctly more than 4,000 malware instances as true positive cases while there are 55 malware instances predicted as benign. In the other side, the model of CNN predicted truly 263 benign instances while the remaining was predicted as malware.

V. Discussion

In this section, we discuss the experimental results and their broader implications. Addressing the malware problem is an ongoing research area. We studied the impact of using both the gray-scale and RGB color images in the malware visualization method. We implemented binary and multi-class classification using both gray-scale imaging and RGB-color images. In our study, we find that converting the binary files to either gray-scale images or RGB color images does not impact the performance results. Hence, focusing on a new approach in feature generation and extraction is required. Using new visualization techniques like Speeded Up Robust Features(SURF), Histogram of Oriented Gradients (HOG), Local Binary Patterns (LBP), and Scale Invariant Feature Transform (SIFT) might better support and enhance the malware detection model [40] but subject to evaluation. In this work, we have included only the GIST-based Image visualization technique for malware detection. Any other visualization and non-visualization-based discussion are out of our work's scope. Nataraj et al. [14], [37], [15]'s original setup included 9,458 samples from 25 malware families with gray-scale image conversion and GIST-based feature extraction. Their approach achieved a classification accuracy score of 98 %. While our results align with the performance results, we extended the framework to include data not only from SARVAM [14] (25 malware families), but also from Xue et al. [18] (10 malware families), and CIC [39]

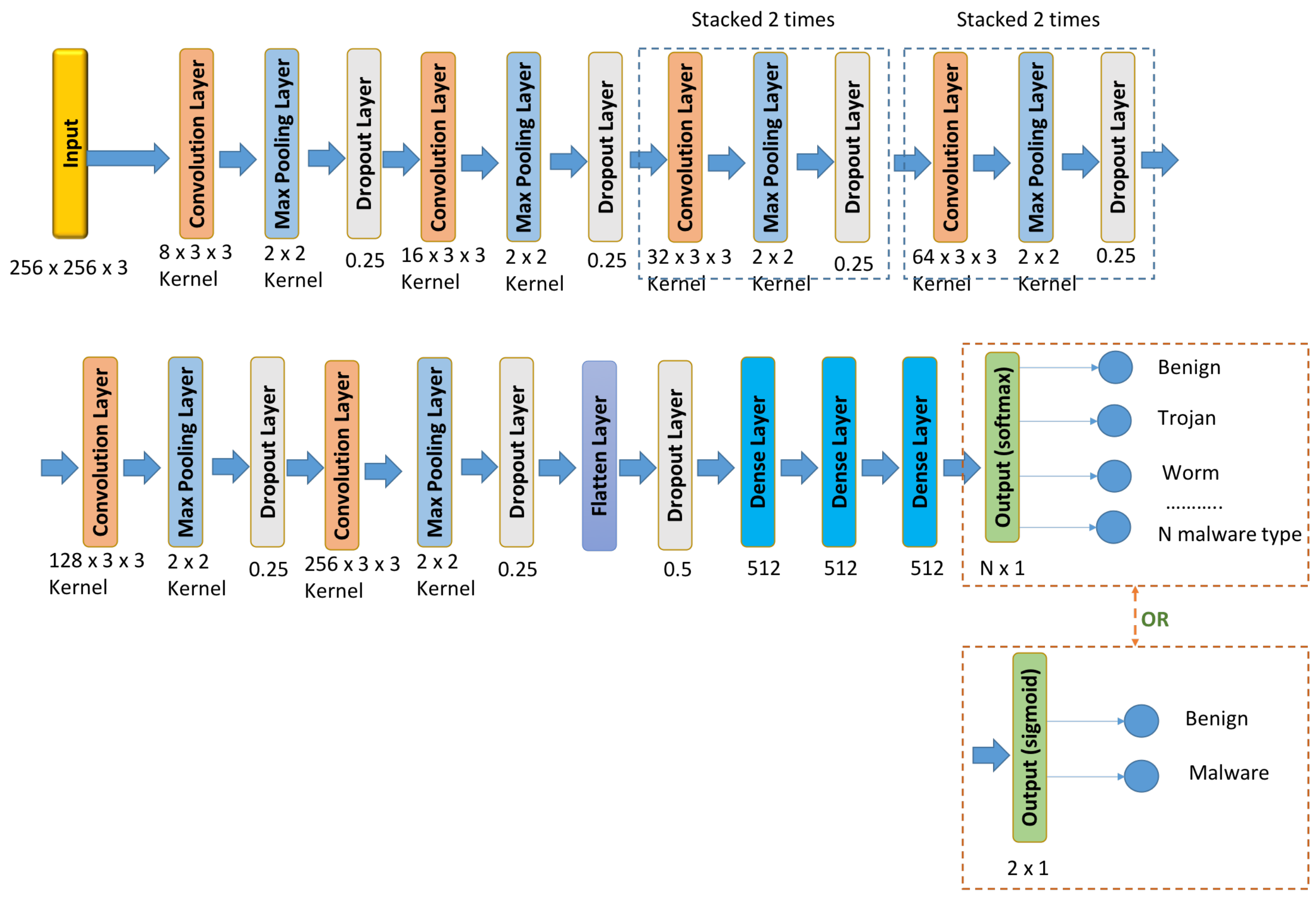


Fig. 10. Convolutional Neural Networks Architecture for Malware Classification.

(6 malware families). In addition, we evaluated the proposed design using various machine learning algorithms i.e. Linear Regression, Random Forest, KNN, CART, SVM. We also have evaluated using CNN-based deep learning technique and MLP. In our setup, we consider the idea of using both the binary and multi-class classifiers that provide handlers to extend the design to an open-set and ensemble-based learners.

Our approach is producing a comparable performance to the above-mentioned classifiers. Even though the datasets used in these classifiers are SARVAM and other datasets, our statistical T-TEST on F1-scores are not statistically significantly different.

VI. Conclusion

In this work, we implement the computer visualization-based technique to build ML-based malware detection. We employ a GIST-based approach to extract malware image features from both the gray-scale and color images of malware binary samples. To study the prediction performance, we empirically analyze various machine learning algorithms. Our experimental study includes the following learning algorithms: linear regression, random forest, k-NN, CART, SVM, and MLP, and a CNN-based deep learning model and selects a candidate learning classifier that can yield better prediction performances. We evaluate our approach using the following malware datasets SARVAM, PhD-thesis, and CIC. In comparison to traditional malware detectors, the visualization-based approach provides a significant performance enhancement. Our study observes that the CNN-based deep-learning model yields significantly better performance when tested against the various malware datasets listed above. Malware authors constantly innovate new methods for implementing sophisticated attacks that make the malware detection as an active research area. Our approach provides a path forward to implement an innovative malware detector. Nevertheless, this research work needs to be verified against various new malware datasets to be more effective. In the future, we would like to extend our research to benchmark datasets and as well conduct a large-scale evaluation.

References

[1] A. Souri and R. Hosseini, “A state-of-the-art survey of malware detection approaches using data mining techniques,” Human-centric Computing and Information Sciences, vol. 8, no. 1, p. 3, 2018.

[2] M. Fredrikson, S. Jha, M. Christodorescu, R. Sailer, and X. Yan, “Synthesizing near-optimal malware specifications from suspicious behaviors,” in 2010 IEEE Symposium on Security and Privacy. IEEE, 2010, pp. 45–60.

[3] X. Hu, T. Chiueh, and K. G. Shin, “Large-scale malware indexing using function-call graphs,” in Proceedings of the 16th ACM conference on Computer and communications security, 2009, pp. 611–620.

[4] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, "Opem: A static-dynamic approach for machine-learning-based malware detection," in International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions. Springer, 2013, pp. 271–280.

[5] H. Sayadi, N. Patel, S. M. PD, A. Sasan, S. Rafatirad, and H. Homayoun, "Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification," in 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). IEEE, 2018, pp. 1–6.

[6] H. Sayadi, H. M. Makrani, O. Randive, S. M. PD, S. Rafatirad, and H. Homayoun, "Customized machine learning-based hardware-assisted malware detection in embedded devices," in 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). IEEE, 2018, pp. 1685–1688.

[7] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics-based detection of android malware through static analysis," in Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014, pp. 576–587.

[8] K. Griffin, S. Schneider, X. Hu, and T. Chiueh, "Automatic generation of string signatures for malware detection," in International workshop on recent advances in intrusion detection. Springer, 2009, pp. 101–120.

[9] D. Venugopal and G. Hu, "Efficient signature based malware detection on mobile devices," Mobile Information Systems, vol. 4, no. 1, pp. 33–49, 2008.

[10] I. Santos, F. Brezo, J. Nieves, Y. K. Penya, B. Sanz, C. Laorden, and P. G. Bringas, "Idea: Opcode-sequence-based malware detection," in International Symposium on Engineering Secure Software and Systems. Springer, 2010, pp. 35–43.

[11] J. Scott, "Signature based malware detection is dead," Institute for Critical Infrastructure Technology, 2017.

[12] A. Pinhero, M. Anupama, P. Vinod, C. Visaggio, N. Aneesh, S. Abhijith, and S. AnanthaKrishnan, "Malware detection employed by visualization and deep neural network," Computers & Security, p. 102247, 2021.

[13] E. M. Rudd, A. Rozsa, M. Günther, and T. E. Boult, "A survey of stealth malware attacks, mitigation measures, and steps toward autonomous open world solutions," IEEE Communications Surveys & Tutorials, vol. 19, no. 2, pp. 1145–1172, 2016.

[14] 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.

[15] L. Nataraj and B. Manjunath, "Spam: Signal processing to analyze malware," IEEE Signal Processing Magazine, vol. 33, no. 2, pp. 105–117, 2016.

[16] K. Han, B. Kang, and E. G. Im, "Malware analysis using visualized image matrices," The Scientific World Journal, vol. 2014, 2014.

[17] K. S. Han, J. H. Lim, B. Kang, and E. G. Im, "Malware analysis using visualized images and entropy graphs," International Journal of Information Security, vol. 14, no. 1, pp. 1–14, 2015.

[18] D. Xue, J. Li, W. Wu, Q. Tian, and J. Wang, "Homology analysis of malware based on ensemble learning and multifeatures," PloS one, vol. 14, no. 8, p. e0211373, 2019.

[19] L. Liu, B. Wang, B. Yu, and Q. Zhong, "Automatic malware classification and new malware detection using machine learning," Frontiers of Information Technology & Electronic Engineering, vol. 18, no. 9, pp. 1336–1347, 2017.

[20] 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.

[21] A. Makandar and A. Patrot, "Malware class recognition using image processing techniques," in 2017 International Conference on Data Management, Analytics and Innovation (ICDMAI). IEEE, 2017, pp. 76–80.

[22] A. Singh, A. Handa, N. Kumar, and S. K. Shukla, "Malware classification using image representation," in International Symposium on Cyber Security Cryptography and Machine Learning. Springer, 2019, pp. 75–92.

[23] B. Zhang, J. Yin, J. Hao, D. Zhang, and S. Wang, "Malicious codes detection based on ensemble learning," in International conference on autonomic and trusted computing. Springer, 2007, pp. 468–477.

[24] E. Menahem, A. Shabtai, L. Rokach, and Y. Elovici, "Improving malware detection by applying multi-inducer ensemble," Computational Statistics & Data Analysis, vol. 53, no. 4, pp. 1483–1494, 2009.

[25] C. Iwendi, S. Khan, J. H. Anajemba, M. Mittal, M. Alenezi, and M. Alazab, "The use of ensemble models for multiple class and binary class classification for improving intrusion detection systems," Sensors, vol. 20, no. 9, p. 2559, 2020.

[26] W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, and T. E. Boult, "Toward open set recognition," IEEE transactions on pattern analysis and machine intelligence, vol. 35, no. 7, pp. 1757–1772, 2012.

[27] J. Henrydoss, S. Cruz, E. M. Rudd, M. Gunther, and T. E. Boult, "Incremental open set intrusion recognition using extreme value machine," in 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, 2017, pp. 1089–1093.

[28] T. Wang and N. Xu, "Malware variants detection based on opcode image recognition in small training set," in 2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA). IEEE, 2017, pp. 328–332.

[29] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware detection with deep neural network using process behavior," in 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), vol. 2. IEEE, 2016, pp. 577–582.

[30] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in Australasian Joint Conference on Artificial Intelligence. Springer, 2016, pp. 137–149.

[31] W. Huang and J. W. Stokes, "Mtnet: a multi-task neural network for dynamic malware classification," in International conference on detection of intrusions and malware, and vulnerability assessment. Springer, 2016, pp. 399–418.

[32] M. Amin, T. A. Tanveer, M. Tehseen, M. Khan, F. A. Khan, and S. Anwar, "Static malware detection and attribution in android byte-code through an end-to-end deep system," Future Generation Computer Systems, vol. 102, pp. 112–126, 2020.

[33] H. Cai, "Assessing and improving malware detection sustainability through app evolution studies," ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 29, no. 2, pp. 1–28, 2020.

[34] B. S. Manjunath and W. Ma, "Texture features for browsing and retrieval of image data," IEEE Transactions on pattern analysis and machine intelligence, vol. 18, no. 8, pp. 837–842, 1996.

[35] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," International journal of computer vision, vol. 42, no. 3, pp. 145–175, 2001.

[36] A. Torralba, K. P. Murphy, W. T. Freeman, M. A. Rubin et al., "Context-based vision system for place and object recognition." in ICCV, vol. 3, 2003, pp. 273–280.

[37] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, 2011, pp. 21–30.

[38] M. Douze, H. Jégou, H. Sandhawalia, L. Amsaleg, and C. Schmid, "Evaluation of gist descriptors for web-scale image search," in Proceedings of the ACM International Conference on Image and Video Retrieval, 2009, pp. 1–8.

[39] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark android malware datasets and classification," in 2018 International Carnahan Conference on Security Technology (ICCST). IEEE, 2018, pp. 1–7.

[40] N. Raj and V. Niar, "Comparison study of algorithms used for feature extraction in facial recognition," Int. J. Comput. Sci. Inf. Technol, vol. 8, no. 2, pp. 163–166, 2017.