

Received October 17, 2021, accepted October 29, 2021, date of publication November 10, 2021, date of current version November 18, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3127195

# Histogram Entropy Representation and Prototype Based Machine Learning Approach for Malware Family Classification

BYUNGHYUN BAEK<sup>1</sup>, SEOUNGYUL EUH<sup>2</sup>, DONGHEON BAEK<sup>3</sup>,  
DONGHOON KIM<sup>4</sup>, AND DOOSUNG HWANG<sup>5</sup>

<sup>1</sup>GI VITA, Seoul 06536, South Korea

<sup>2</sup>Security Technology Institute, KSign, Seoul 06231, South Korea

<sup>3</sup>Department of Oral Microbiology and Immunology, Dankook University, Cheonan 31116, South Korea

<sup>4</sup>Department of Computer Science, Arkansas State University, Jonesboro, AR 72467, USA

<sup>5</sup>Department of Software Science, Dankook University, Yongin-si 16890, South Korea

Corresponding author: Doosung Hwang (dshwang@dankook.ac.kr)


This work was supported by the Institute for Information and Communications Technology Promotion (IITP) funded by the Government of Korea (Ministry of Science and ICT (MSIT), Trust-based cyber security platform for smart facility environment) under Grant 2019-0-00197.

**ABSTRACT** The number of malware has steadily increased as malware spread and evasion techniques have advanced. Machine learning has contributed to making malware analysis more efficient by detecting various behavioral and evasion patterns. However, when analyzing large-scale malware datasets, malware analysis through learning models has both high temporal and spatial complexity. In order to address these problems, this work proposes a low-dimensional feature using histogram entropy and a prototype selection algorithm using hyperrectangles. The low-dimensional feature forms an  $L \times 256$  map according to the preselected parameter  $L$ . The prototype selection algorithm divides the input space into overlapping subspaces where each subspace is decided by its hyperrectangle that becomes a prototype in the same class. A set cover optimization algorithm is employed to select a small number of prototypes that construct a new training dataset. A set of prototypes selected by the prototype selection algorithm is used to classify malware families. The experiment compares the performance of machine learning models for the histogram entropy feature using both the BIG 2015 dataset and the collected dataset. The integrated approach is evaluated using learning algorithms, such as Decision Tree, Random Forest, XGBoost, and CNN. The experimental results indicate that learning models perform competitively when compared to the entire dataset, while the proposed selection approach benefits from smaller datasets and lower time complexity.

**INDEX TERMS** Malware family classification, histogram entropy, low-dimensional feature, hyperrectangle, prototype selection, ensemble model, machine learning.

## I. INTRODUCTION

Malware is software that is installed silently and secretly on computers, servers, clients, and networks to perform actions that users do not expect. Computers connected to the network are more likely to spread malware and pose a significant threat to the advancement of information and communication technologies. Malware that has recently been discovered is spreading out through its own evasion technology as well as advanced vulnerability analysis technology. The detection and response to new or modified malware is critical

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

to the advancement of information technology, and ongoing research and improvement efforts are required.

Malware with evasion technology can be detected through continuous monitoring, but it takes an inordinate amount of time and effort to execute and analyze dubious executables. Furthermore, it is difficult to define rules of malicious behavior, and lower high false positive detection rates. Malware detection technologies based on machine learning have been investigated in order to address these drawbacks [1]–[3].

Machine learning for malware detection explores classification rules based on feature vectors or employs similarity based metrics for classifiers. In general, classification prediction is accomplished through the learning process by

discovering hidden pattern rules. Such detection methods can also distinguish intrinsic but hidden patterns among benign and malware. However, in order to ensure a robust analysis using machine learning, a sufficient number of training examples must be collected. For malware detection systems, feature extraction methods such as opcode (operation code) [4], [5], function call graph [6], [7], string signature [8], entropy [9] and byte  $n$ -gram [10], [11] have been studied. Low-dimensional features has been studied because features with high-dimensional presentation need a huge amount of training time.

Malware developers are constantly creating new malware or employing variant technologies to avoid detection by anti-virus software. As a result, the number of malware families grows year after year, raising issues that increase time complexity and space complexity to analyze malware variants. In the case of a large amount of training data, there is a high possibility of redundant and noisy data, which increases the complexity of learning model, and a large amount of training time is required [12], [13]. These issues have been targeted to select a small set of prototypes which can replace the original dataset [13], [14]. Therefore, classification based on a set of prototypes can achieve comparable performance to the entire dataset while eliminating inessential data and reducing time complexity [15], [16].

Prototype selection methods employ similarity metrics and class labels from the dataset [14], [17]. The similarity among instances is measured using Euclidean distance, Manhattan distance, Mahalanobis distance, and so on [18]. A prototype represents a subset of instances that are placed at a constant distance in the same class. A selected prototype covers as many instances of the same class as possible and becomes a new training instance for classification models. Previous research has employed hyperspheres [15], [19], [20], and hyperrectangles [21] as prototype selection approaches to divide multidimensional space into subspaces and select a small subset of instances to replace the entire dataset.

This study proposes a low-dimensional feature representation of fixed size based on histogram entropy, as well as a prototype selection method for large-scale malware datasets based on hyperrectangles. The contribution of this study is as follows:

- Two dimensional (2D) histogram entropy map is designed to characterize malware for statistical analysis. The feature is a low-dimensional feature extraction method based on entropy information and a fixed size.
- A prototype selection method is proposed on the basis of hyperrectangles that select a small set of prototypes which machine learning algorithms can learn instead of the entire dataset.
- The process of extracting features can be visualized to identify key patterns for malware detections.
- Experimental results show that it provides comparable performance for machine learning algorithms with only

a relatively small new dataset generated from the entire dataset using our prototype selection method.

This paper is organized as follows. Section II discusses related work. Section III addresses data collection and feature extraction methods. Section III-D proposes the prototype selection algorithm. Section IV evaluates learning models for identifying malware. Finally, Section V concludes this paper with future works.

## II. RELATED WORK

Feature engineering for malware static analysis makes use of opcode or byte data from executable binaries. Feature extraction using opcode must include a disassembly phase, but this phase has the limitation that packed and obfuscated parts may result in invalid and incorrectly assembled code [2], [3]. As another way, an entropy based feature representation has been chosen to quantitatively compare the entire structure of malware at the byte level [22]–[27].

Various representations of static features have been proposed for malware detection:  $n$ -gram byte feature, entropy or hashing feature for binaries and  $n$ -gram opcode feature, DLL call or API call graph feature from assembly code, etc. However, if the feature space of these malware becomes too large, the feature vector size will change. This tends to make feature engineering more difficult. Efforts have been made to convert malware into fixed size data to make malware features robust. Examples include a 2D grayscale image, window entropy map [27], histogram entropy map [23], and hashing based map [28]. The values calculated by applying a sliding window over an executable file were integrated to represent malware features.

Table 1 summarizes the studied approaches including classification type (Class), feature type, detection model and the details on the datasets in use. The datasets are federated from the known datasets or self-collected datasets. For example, there are Microsoft Malware Classification Challenge (BIG 2015, [29]), Malica-project [30], Virus Total [31], Vx Heaven [32], VIPRE [33], MalImg [34], Malwares [35], etc. The classification type is defined as a binary classification for malware detection or as a  $k$ -class classification for malware family identification. The feature type is categorized into one of data, entropy, or image driven feature engineering.

Data driven feature engineering was studied by Burnap *et al.* [36] and Fan *et al.* [37]. A dynamic analysis of the data collected through the Cuckoo sandbox [43] was performed, and the feature vectors were made up of file access log, registry key access, process execution, packet log, and usage patterns of CPU and memory [36]. There have also been studies that divide the data into a certain number of chunks and run binary codes using clustering and classification to analyze it dynamically in order to reduce overhead with a large amount of malware data [41]. The winner of the SOFM (Self-Organizing Feature Map) model of the input feature vector was selected, and its class was predicted with the closest class to the BMU (Best Matching Unit)

TABLE 1. A summary of studied malware detection systems.

Studied	Class	Total	Malware	Benign	Feature Type			Model	Dataset
					Data	Entropy	Image		
Burnap [36]	2	1,188	594	594	✓			SOFM+1-NN	Virus Total [31]
Fan [37]	2	10,307	8,847	1,460	✓			Neural network	Vx Heaven [32]
Yuxin [38]	2	9,200	4,600	4,600	✓			DBN	BIG 2015 [29] et al.
Lyda [9]	-	21,576	-	-		✓		Confidence interval analysis	-
Han [39]	50	1,000	1,000	-		✓		Histogram analysis	Vx Heaven [32]
Nataraj [22]	25	9,458	9,458	-			✓	GIST+k-NN	MallImg [34]
Gibert [25]	9	10,868	10,868	-			✓	CNN	BIG 2015 [29]
Luo [40]	9	10,868	10,868	-			✓	CNN	BIG 2015 [29], MallImg [34]
Ni [28]	9	10,868	10,868	-	✓		✓	CNN	BIG 2015 [29]
Ahmadi [24]	9	10,868	10,868	-	✓	✓	✓	XGBoost	BIG 2015 [29]
Saxe [23]	2	431,926	350,016	81,910	✓	✓	✓	CNN	Virus Total [31]
Dey [26]	9	10,868	10,868	-	✓	✓	✓	k-NN	BIG 2015 [29]
Rieck [41]	24	33,698	33,698	-	✓			Nearest prototype classifier	VIPRE [33]
Hu [42]	20	132,234	132,234	-	✓			Prototype-based clustering	Vx Heaven [32]
Euh [27]	2	122,963	102,963	20,000	✓	✓	✓	Ensemble	Malwares [35]

through the Euclidean distance. In case of multiple winners, the predictive class with the highest frequency was selected. The highest accuracy of 90% was demonstrated when the size of the feature map was set to  $80 \times 80$ . Fan *et al.* [37] proposed the pattern mining approach to codify malware feature vectors and All-Nearest-Neighbor (ANN) algorithm to detect malware. In addition, features were reselected by applying the MIE (Malicious Instruction Extraction) technique to remove unnecessary information and their detection rate was about 96.0%. Yuxin and Siyi [38] adopted a deep belief network to encode opcode sequences and applied the backpropagation algorithm to learn the final classifier. Their feature was coded by generating 3-gram according to the control flow analysis of opcode sequences. The experiment yielded about 98.0% accuracy.

Researchers investigated entropy driven detection because malware identification becomes difficult due to encryption, packing, obfuscation and polymorphism [9], [39], [44]. The entropy values of malware belonging to different malware families tend to differ significantly. Lyda and Hamrock [9] suggested the entropy analysis method by examining the statistical difference among executables. They utilized the confidence-interval based method by calculating the amount of statistical variation of bytes in a data stream and summing the frequency of each observed byte value in a fixed length data block. They found that higher entropy values tend to correlate with the presence of encryption or packing.

Sorokin [44] proposed the structural entropy approach divided files into segments: executable code, text, and packed area. Each segment was characterized in terms of size and homogeneity by entropy information. First, the wavelet analysis was used to divide the file into segment sequences of varying entropy levels. The next step detected malware by calculating the Levenshtein distance between segment sequences to determine the degree of similarity. Han *et al.* [39] converted PE (Portable Executable) files into bitmap images and compared the entropy changing tendency. Their analysis identified the malware family by comparing the similarity of both two entropy graphs of the test malware

and of the previously known malware family. The database consisted of 1,000 malware of 50 families from Vx Heaven and an accuracy was approximately 98.0% when the threshold was 0.75.

Nataraj *et al.* [22] reshaped malware binaries into 8-bit grayscale images based on their file size range. The grayscale image was converted into GIST feature vectors by using the Gabor filter to compute local feature maps. All of the local feature maps were combined into a single GIST feature, which was then downsampled to a fixed size training instance. Using  $k$ -NN among MallImg's GIST feature vectors, they reported the detection rate of 97.2% for malware family identification. Llaurodó *et al.* [25] employed a CNN (Convolutional Neural Network) model with 2D grayscale images of BIG 2015 and MallImg datasets. The input size for MallImg was  $256 \times 256$ , and the input size for BIG 2015 was  $128 \times 128$ , with their dimensions adjusted by experiments to learn the CNN model with fixed length. For 5-fold and 10-fold cross validation, the experimental results achieved accuracy of 97.3% and 97.5%, respectively. Luo and Lo [40] also created the MallImg and BIG 2015 training datasets using the local binary pattern (LBP) and tested a CNN detection model. Their CNN model demonstrated that the LBP feature outperformed the GIST feature, with accuracy of 93.92% for MallImg and 93.57% for BIG 2015. Using both Haralick and LBP to Nataraj's grayscale image, Ahmadi *et al.* [24] generates 2D images from BIG 2015. Their XGBoost model yielded an average of 95.0% for both training vector and the detection model in 5-way cross-validation.

Ni *et al.* [28] proposed the MCSC (Malware Classification using SimHash and CNN) approach. They decreased feature extraction time by selecting the main blocks only because it took a long time to extract all opcode as features. The main code block tends to include malware behavior information as well as the CALL instruction. The opcode sequence differs depending on the size of the malware file and is hashed to generate a binary vector of the fixed size. Thus, the sum of weights of all binary vectors in the sequence is calculated, and the weight sum vector is converted into a  $16 \times 16$  image.

**TABLE 2.** Malware families in BIG 2015.

Family name	No. of data	Ratio (%)	Type
Ramnit	1,541	14.2	Worm
Lollipop	2,478	22.8	Adware
Kelihos_ver3	2,942	27.1	Backdoor
Vundo	475	4.4	Trojan
Simda	42	0.4	Backdoor
Tracur	751	6.9	TrojanDownloader
Kelihos_ver1	398	3.7	Backdoor
Obfuscator.ACY	1,228	11.3	Obfuscated malware
Gatak	1,013	9.3	Backdoor
Total	10,868	100.0	

The MCSC performance reported an accuracy of about 87.0% for the CNN model with BIG 2015.

Dey *et al.* [26] proposed a detection method for improving Natarj's image driven algorithm with entropy filtering for 2D image transformation [22]. The variants of metamorphic engines can avoid detection by anti-virus programs based on signatures and primary obfuscation techniques that disguise malicious commands. This method, however, leaves suspicious patterns at the bit level. This can be identified through entropy calculations. The local entropy value of the gray image determines the structure of entropy filtering in response to an entropy image. The  $k$ -NN classifier experiment produced slightly better results than Natarj's method [22].

Hu *et al.* [42] used opcode to do static analysis to compensate for the limitation of dynamic analysis, called MutantX-S. Their static-feature-based approaches are far more scalable than their dynamic-feature-based approaches. They converted malware binaries into an opcode sequence, allowing  $n$ -gram features to be extracted more quickly. With a linkage clustering and a prototype-based nearest classification [41], Rieck *et al.* addressed the scalability issues in terms of run-time performance and memory requirement. Their incremental approach was proposed for behavior-based analysis of malware classifications, which could handle the behavior of thousands of malware per day.

Multiple features for malware detection in Ahmadi *et al.* [24], Saxe and Berlin [23], and Euh *et al.* [27] were proposed. These features were built using data, entropy, and images. Ahmadi *et al.* [24] proposed the malware family detection with combined features from hex dump-based features, assembled code features and entropy images. They applied the XGBoost [45] to BIG 2015 through 5-way cross-validation. Each independent feature demonstrated 75.6% to 99.1% accuracy, and the entire collection of features, including the entropy feature, demonstrated approximately 99.8% accuracy. Saxe *et al.* [23] designed a four-layer neural network ( $1024 \times 1024 \times 1024 \times 1$ ) to detect malware and benign. The final feature was composed of byte entropy histogram, PE import and meta-data, and string data. Their prediction results of the learned model were calibrated through the Bayesian method. The detection rate was 95.0% for all the integrated feature vectors of the prepared PE Import, byte

**TABLE 3.** The collected data from Malwares.com.

Type	No. of data	Ratio (%)
Malware	65,704	76.7
Benign	20,000	23.3
Total	85,704	100.0

entropy, metadata, and strings. Euh *et al.* [27] employed tree ensemble models for 2-gram, gram matrix, WEM (Window Entropy Map), API-DLL, and API from executable and disassembled files. Their features were designed to reduce the original feature dimensionality and decreased the time complexity of ensemble models. For each proposed feature, they compared the performance of AdaBoost, XGBoost, Random Forest, Extra Trees, and Rotation Trees. WEM's XGBoost performed best with 98.0% in terms of accuracy and AUC-PRC evaluation.

### III. RESEARCH APPROACH

#### A. MALWARE DATASET

Our proposed method is evaluated with BIG 2015 and the Malwares dataset, where the total size is about 115 GB. Each instance includes its own assembly code and binary file. Table 2 and 3 show the number of data and information on the test datasets. Each malware family contains at least 42 (0.4%) of instances and up to 2,942 (27.1%) of instances (Table 2). The dataset for malware and benign classifications was collected from Malwares.com [35]. The number of malware is 65,704 (76.7%) and the number of benign is 20,000 (23.3%) (Table 3). The benign dataset is also used for a malware detection problem with the BIG 2015 dataset.

#### B. HISTOGRAM ENTROPY

As malware vectorization, a 2-gram feature showed excellent performance, but has a tendency of high dimensional elements to represent a single malware [24], [27], [38]. If the data dimension increases, the input space increases proportionally, resulting in a sparse distribution. Additionally, the number of model parameters increases and a training dataset should consist of sufficient instances in order to construct a robust learning model. We design a low-dimensional feature using histogram entropy information of byte sequences. Fixed length and low-dimensional malware vectorization takes advantage of reducing training model complexity, preventing overfitting, and expecting high generalization performance.

Figure 1 illustrates the process of generating our histogram entropy feature from an executable through applying a sliding window and computing histogram frequency and entropy. Figure 1 (a) is the 2D image of an Obfuscator.ACY instance which is shaped with  $N \times L$  through applying sliding window size  $L$  and stride size  $s$ . Figure 1 (b) is the same representation of Figure 1 (a) in hexadecimal. The actual size of the input image is 1,469,952  $\times$  1,024. The  $k^{\text{th}}$  window is

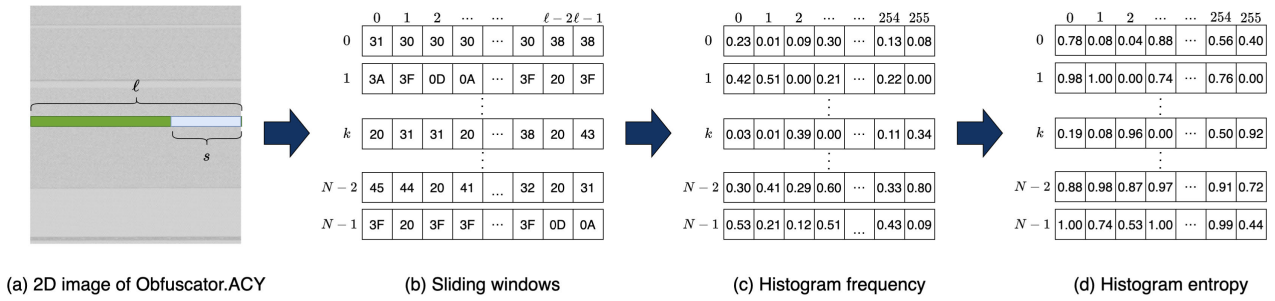


FIGURE 1. Steps from 2D image to entropy histogram.

represented by vector  $w_k$ .

$$w_k = [b_0^{(k)}, b_1^{(k)}, \dots, b_{L-1}^{(k)}]$$

for  $k = 0, 1, \dots, N-1$  and  $b_j^{(k)}$  stands for a byte value between 0 and 255. When  $L = 1, 024$  and  $s = 256$ , Figure 1 (c) is the normalized byte histogram transformation of Figure 1 (b). The histogram vector  $h_k$  of the  $k^{\text{th}}$  window  $w_k$  becomes

$$h_k = [h_0^{(k)}, h_1^{(k)}, \dots, h_{255}^{(k)}],$$

where  $h_j^{(k)}$  is the  $j^{\text{th}}$  bin value. The histogram entropy vector  $e_k$  of  $w_k$  is

$$e_k = [e_0^{(k)}, e_1^{(k)}, \dots, e_{255}^{(k)}].$$

The bin entropy  $e_j^{(k)}$  of the  $j^{\text{th}}$  bin of the  $k^{\text{th}}$  window is calculated by the Shannon entropy.

$$e_j^{(k)} = -h_j^{(k)} \log_2 h_j^{(k)} - (1 - h_j^{(k)}) \log_2 (1 - h_j^{(k)})$$

Every bin entropy  $e_j^{(k)}$  is associated with a state of uncertainty in terms of two discrete probabilities. If  $h_j^{(k)}$  is close to 0 or 1, then  $e_j^{(k)}$  is a lower value. However, if  $h_j^{(k)}$  approaches 0.5, then  $e_j^{(k)}$  shows a high value. The value of  $e_j^{(k)}$  ranges over  $[0, 1]$ .

Figure 2 illustrates the process to construct a fixed length feature map. The horizontal direction of a feature map  $M$  goes along histogram bin indexes and the vertical direction indicates  $L$  discretized values:  $M = [m_{ij}]_{L \times 256}$  for a small value  $\delta$ .

$$\text{loc}(e_j^{(k)}) = (l, j) \quad \text{if } e_j^{(k)} \in (\delta(l-1), \delta l]$$

$$m_{ij} = \sum_{k=0}^{N-1} \sum_{i=0}^{255} 1[\text{loc}(e_j^{(k)}) = (l, j)] e_i^{(k)},$$

where  $l = 1, 2, \dots, L$  and  $1[x]$  is 1 if  $x$  is true or 0 otherwise.  $\text{loc}(e_j^{(k)})$  returns the coordinates where  $e_j^{(k)}$  will be added. Therefore, the malware representation becomes a 2D array.

Figure 2 is an example of  $L = 10$  and  $\delta = 0.1$ , and the y-axis is divided into 10 steps:  $(0, 0.1]$  for level 1,  $(0.1, 0.2]$  for level 2, and  $(0.9, 1.0]$  for level 10. In the right figure, a place labeled  $\star$  sums two inputs of 0.81 and 0.87 resulting in 1.68. This is because 0.81 and 0.87 belong to section

$(0.8, 0.9]$  whose level is  $l = 9$ . The coordinates of each element appearing for all  $e_j^{(k)}$  are calculated and accumulated to the proper cell over  $M$ .

$M$  represents the degree of uncertainty accumulated in bin (horizontal directions) and  $L$  (vertical direction) to construct a 2D map of an executable file with a fixed size. In addition, the distribution of the vertical direction expresses the change by level on the horizontal direction. A fixed size feature configuration is required and a preemptive condition for applying various machine learning algorithms.

### C. HISTOGRAM ENTROPY VISUALIZATION

Visualization provides one way to identify key patterns in analyzing malware. The analysis phase considers the number, location, and shape of peaks appearing on the histogram and place high weights on the largest peaks [46]. Figure 3 and 4 shows the examples of the proposed histogram entropy method which are generated from BIG 2015. The entropy datagram with  $256 \times L$  was created by setting  $L = 1$ , and the y-axis represents the summed entropy value over bins. Histogram entropy depicts different patterns which is able to determine the malware families.

Figure 3 (a) and (b) are Gatak examples, where the change of entropy values is generally lower than 0.4 and peaks of values are intermittently visible. Figure 3 (c) to (f) are the examples of Kelihos. They show various patterns depending on the version, repeatedly display a pattern that occurs lower than 0.4 in case of version 1, and peaks above 0.8 are seen before the 20<sup>th</sup> bin. The example of version 3, on the other hand, has zigzag patterns throughout, with peaks above 0.6 in the second half.

Figure 3 (g) and (h) are the examples of Lollipop. They have a number of peaks above 0.6 before the 50<sup>th</sup> bin, and the remaining part retains entropy values of 0.2 or less. Figure 3 (i) and (j) are the examples of Obfuscator.ACY. There is a continuous and distinct change in entropy histogram according to the x-axis. They decrease at first, then increases in the middle, and tend to decrease gradually afterward. It is evident that the changes between neighboring bins gradually decrease or increase.

Figure 4 (a) and (b) are the examples of Ramnit. They show a similar pattern to Gatak, but a value greater than

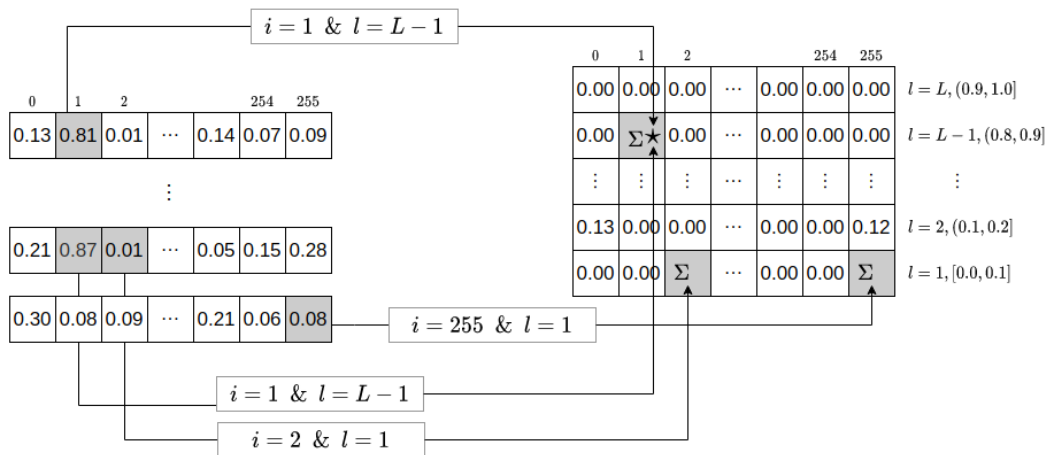


FIGURE 2. Generation of the fixed 2D entropy feature.

0.1 appears overall, and high peaks are greater than 0.4. Figure 4 (c) and (d) show the examples of Simda which are composed of values below 0.2. There are repetitive patterns that gradually appear after increasing. Figure 4 (e) and (f) are the examples of Tracur. They have a entropy value of 0.4 or more in bins 40 to 60, and other parts are composed of values less than 0.2. Figure 4 (g) and (h) are the examples of Vundo. They have a tendency to rise and drop rapidly in the range of 0.0 and 0.3.

The histogram changes in the same malware family appear with its own unique patterns and these patterns are very similar. This is because this change is reinforced when the degree of disorder of Shannon entropy is high, and the change in low entropy is poorly expressed. The entropy change of Obfuscator is distinct from other malware families, and the change of entropy value is expressed at a high level. That is, the change in entropy on the  $x$ -axis is analyzed in the form of continuously increasing or decreasing, and high peak values do not appear. In the pattern of other malware families, high peak values appear repeatedly, and there are patterns showing low changes in other parts. Simba and Vundo show similar patterns, but Vundo displays repeated and fluctuated patterns within some range.

#### D. PROTOTYPE SELECTION APPROACH

To solve problems arising from learning through large-scale malware, we propose a prototype selection algorithm via building hyperrectangles. A hyperrectangle is determined as a partial area within the homogeneous class distribution and includes the same class instance. The selected set of prototypes preserves the class distributions and constructs a new training dataset.

##### 1) PROBLEM FORMALIZATION

A  $c$ -classification problem is defined with  $D = \{(x_i, y_i) | i = 1, \dots, N\}$  where  $x^i \in \mathbb{R}^d$  and  $y_i \in \{1, \dots, c\}$ . The dataset of class  $l$  becomes  $D_l = \{(x_i, l) | i = 1, \dots, n_l\}$ , and the size of  $D_l$  is  $N_l = |D_l|$ . Thus, the size of the entire dataset is

$\sum_{l=1}^c N_l = N$ . The hyperrectangle based prototype selection algorithm (PSA) takes  $D$  as an input and yields  $P = \{P_l | l = 1, \dots, c\}$  as a prototype dataset, where  $|P_l| \ll N_l$  for  $l = 1, 2, \dots, c$ .

##### 2) HYPERRECTANGLES EMBEDDING HOMOGENEOUS DATA

A hyperrectangle takes an area of the input space which includes some training instances in  $D$ . A hyperrectangle is usually defined with  $d$  coordinate points. Alternatively, a hyperrectangle  $h$  is represented only with the maximum and minimum coordinates and instance index set:  $h = \langle h^{\max}, h^{\min}, I \rangle$ . The distance between  $x \in \mathbb{R}^d$  and  $h$  is calculated by Equation (1).

$$\text{dist}(x, h) = \max_{i \in [1, d]} \{|x - \text{mid}| - r\} \quad (1)$$

Here,  $\text{mid} = 0.5(h^{\max} + h^{\min})$  and  $r = 0.5(h^{\max} - h^{\min})$ . The index of  $x$  is appended to  $I$  if  $\text{dist}(x, h) \leq 0$ . Otherwise,  $x$  exists out of the hyperrectangle  $h$ .  $\text{dist}(x, h)$  becomes a distance measure that determines whether  $x$  is located within the input space represented by  $h$ .

Hyperrectangles separate the input space into smaller regions, where each region contains some instances within the same class. Two hyperrectangles can overlap or include the same instances. Let  $s(h|D)$  stand for a covering set of a hyperrectangle  $h$  from  $D$ .

$$s(h|D) = \{z | \text{dist}(z, h) \leq 0 \text{ and } z \in D\} \quad (2)$$

Figure 5 shows a contour plot for the Equation (1) appearing in the range  $[-0.5, 0.5]$ , assuming that  $h$  is the minimum point  $h^{\min} = (-0.15, -0.15)$  and maximum point  $h^{\max} = (0.15, 0.15)$ . The size of hyperrectangle  $h$  is determined by the diagonal length between  $h^{\min}$  and  $h^{\max}$ , and the parameter  $\theta$  is defined to adjust the increasing size of  $h$ . A hyperrectangle  $h$  extends itself by adding an instance  $x$  if  $\text{dist}(x, h) \leq \theta$  and their class is the same. Through constructing hyperrectangles from  $D$  with  $\theta$ , the hyperrectangle set  $H$  is acquired as follows.

$$H = \{h_i = \langle h_i^{\max}, h_i^{\min}, I_i \rangle | i = 1, \dots, N\}, \quad (3)$$

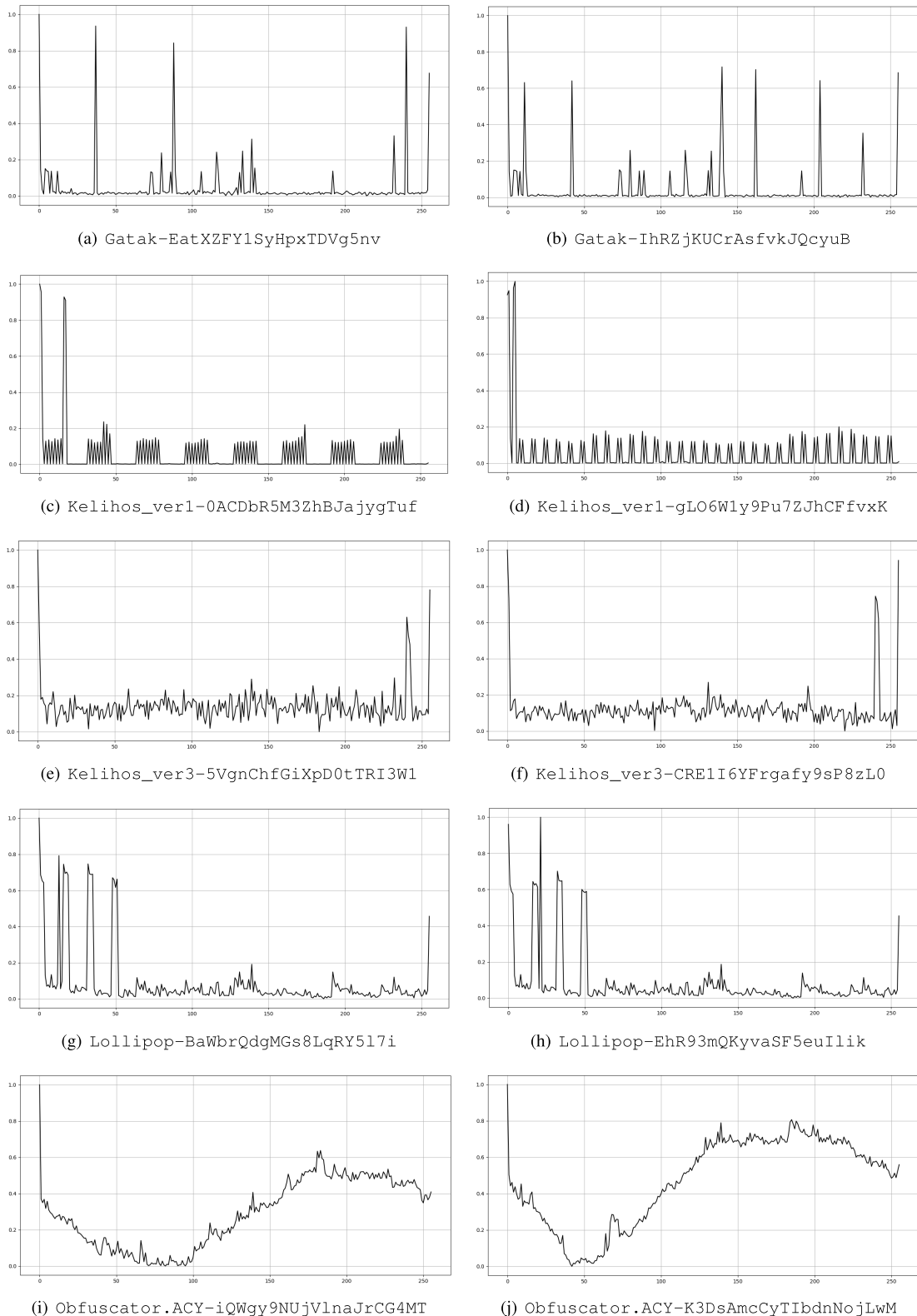


FIGURE 3. Examples of entropy histogram visualization.

where  $I_i$  is the index set of instances that  $h_i$  covers and  $h_i^{\max}$  (or  $h_i^{\min}$ ) is the element-wise maximum (or minimum) coordinates for all instances in  $I_i$ .

### 3) PROTOTYPE SELECTION ALGORITHM

The solution of PSA is to find a small set  $H_{\text{opt}}$  from  $H$  ( $|H_{\text{opt}}| \ll |H|$ ) satisfying  $D = \cup_{h \in H_{\text{opt}}} s(h|D)$ . After

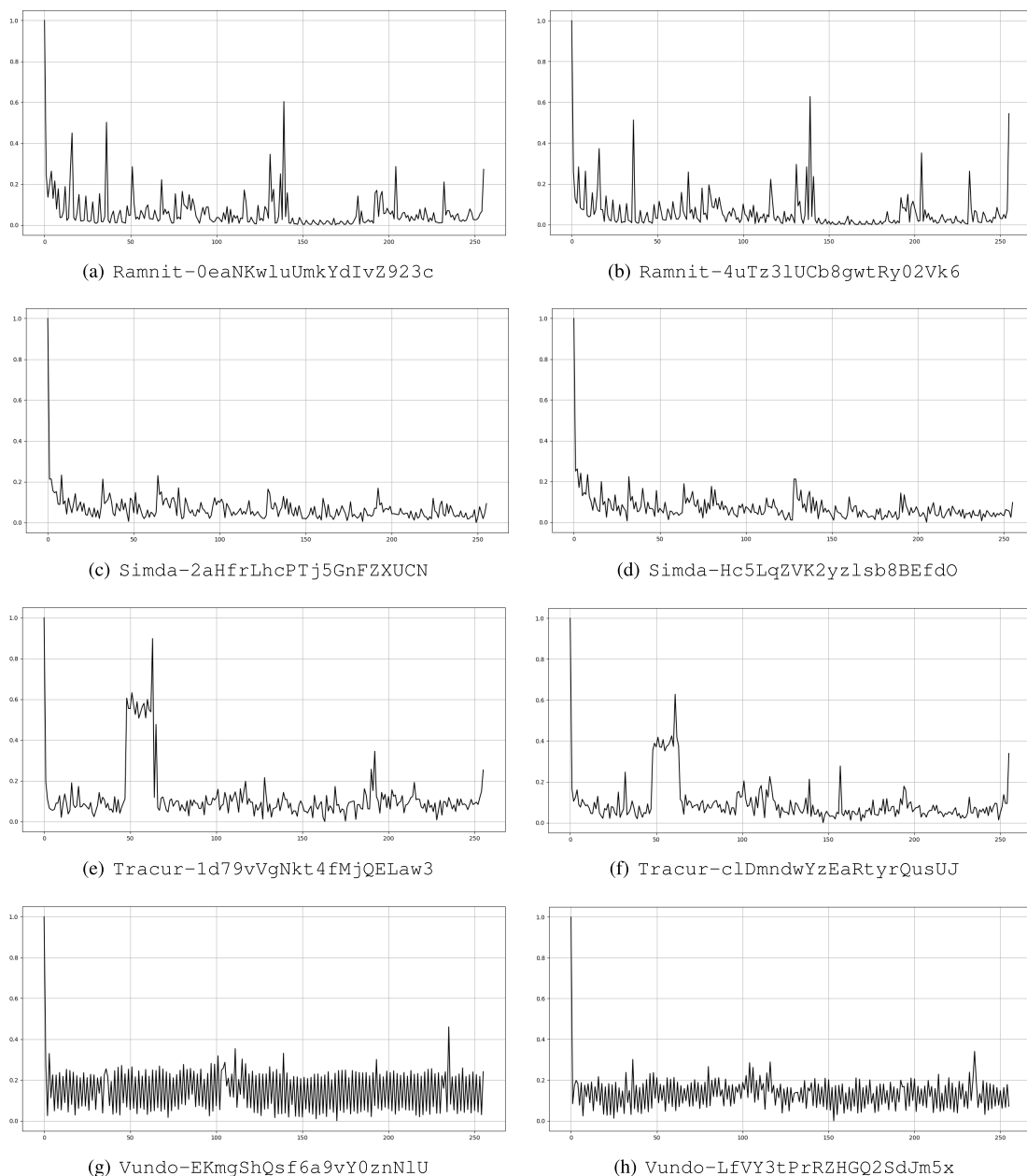


FIGURE 4. Examples of entropy histogram visualization.

generating  $H$ ,  $H_{opt}$  becomes the solution of a set covering problem [47]. So, the greedy approach is generally chosen to find the solution  $H_{opt}$ .

For given  $D$  and  $\theta$ , instead of generating  $H$ , an improved greedy method of finding a solution is adopted by constructing hyperrectangles one by one. PSA gradually approach the final solution  $H_{opt}$  through random selection. Randomly selected instance expands its coverage area while finding and storing instance indexes of the same class included within the distance according to Equation (1).

Algorithm 1 is the pseudocode of PSA. The input parameters in  $PSA(D, \theta)$  are the training dataset  $D$  and parameter  $\theta$ ,

and the output is the set of hyperrectangles. A random number is used to shuffle the order of the instances in  $D$ .  $H$  denotes the set of hyperrectangles to be constructed as a solution and is initially empty.  $C$  is the set of instance indexes existing in the hyperrectangle set  $H$ . If  $i \in C$ ,  $(x_i, y_i) \in D$  has already been covered by a certain  $h \in H$ . Initially,  $C$  is the empty set.

The outer loop selects a candidate hyperrectangle index and the instance indexes that  $h \in H$  covers are added one by one at the inner loop. If the index  $i$  of  $(x_i, y_i) \in D$  of the outer loop does not belong to  $C$ , the inner loop starts to compose a new hyperrectangle  $h$ . A hyperrectangle  $h = \langle h_i^{max}, h_i^{min}, I_i \rangle$  is initialized from  $(x_i, y_i)$ , where  $I_i$  includes



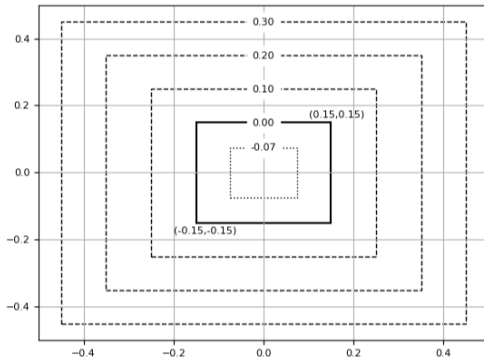


FIGURE 5. Contour plot for Equation (1).

#### Algorithm 1 Prototype Selection Algorithm (PSA)

```

1: procedure PSA(D,  $\theta$ )
2:    $D \leftarrow$  Shuffle D randomly
3:    $H \leftarrow \phi$ 
4:    $C \leftarrow \phi$ 
5:   for  $(x_i, y_i) \in D$  do
6:     if  $i \notin C$  then
7:       // Generate a new hyperrectangle
8:        $h \leftarrow \langle h_i^{\max}, h_i^{\min}, I_i \rangle$ 
9:        $h_i^{\max} \leftarrow x_i$ ;  $h_i^{\min} \leftarrow x_i$ ;  $I_i \leftarrow \{i\}$ 
10:      for  $(x_j, y_j) \in D$  and  $i \neq j$  do
11:        if  $y_j = y_i$  and  $j \notin C$  and  $\text{dist}(x_j, h) \leq \theta$  then
12:          // Add a new member to a hyperrectangle
13:           $h_i^{\max} = \text{ele\_wise\_max}(h_i^{\max}, x_j)$ 
14:           $h_i^{\min} = \text{ele\_wise\_min}(h_i^{\min}, x_j)$ 
15:           $I_i \leftarrow I_i \cup \{j\}$ 
16:        end if
17:      end for
18:       $H \leftarrow H \cup \{ \langle h_i^{\max}, h_i^{\min}, I_i \rangle \}$ 
19:       $C \leftarrow C \cup I_i$ 
20:    end if
21:  end for
22:  return H
23: end procedure

```

the index  $i$ . The inner loop expands the coverage area of  $h$  by searching for a new  $(x_j, y_j)$  that has not been included yet. The selected  $(x_j, y_j)$  is  $j \notin C$  and satisfies  $\text{dist}(x_j, h) \leq \theta$ , and at the same time, update  $h_i^{\max}$  and  $h_i^{\min}$  through element-wise operations. An a new element of  $h$ , the index  $j$  is added to  $I_i$ .

When  $h$  is created from all  $j \notin C$  and  $i \neq j$  that have not been covered yet, the inner loop terminates, and  $h$  becomes a member of  $H$ , and every element of  $I_i$  is included to  $C$ . The same process is repeated for the selected instances in the next outer loop. If the size of  $C$  is equal to  $|D|$ , the algorithm returns  $H$  as the final solution.

A new training dataset is generated from  $H$ . A training instance of  $h = \langle h^{\max}, h^{\min}, I \rangle \in H$  is considered as the mean or median of all instances in  $I$ . New instances by the mean divide the sum of all instances in  $h$  by the number

of elements. Meanwhile, the median of  $h$  is the coordinate average of  $h^{\max}$  and  $h^{\min}$ . A new instance has a one-to-one correspondence to its own hyperrectangle and can be not placed outside the subregion by  $h$ . Therefore, the distribution of the new dataset created by  $H$  is comparable to that of the original dataset. In addition, the class boundaries induced by a machine learning algorithm from the new dataset become similar to those learned from the original dataset.

By dividing the input data space via hyperrectangles, a small number of new training data can be generated while maintaining the distribution of class data. The total number of new training instances is equal to the number of elements in  $H$ . Moreover, the size of the new training dataset is affected by  $\theta$ . When class instances are mixed and distributed, the number of selected hyperrectangles increases, whereas when class areas are kept isolated, the number of hyperrectangles tends to decrease. The preselected  $\theta$  is also a factor in determining the number of selected hyperrectangles and their coverage areas.

#### 4) ALGORITHM COMPARISON

We compare and analyze the PSA performance with prototype selection algorithms using hyperspheres. Interpretable prototype selection [15] (IPS) constructs a hyperspheres that divide class areas using a distance measure and a fixed radius. PSA proposes an optimization technique for selecting a small number of prototypes containing all possible training data. The technique employs a stepwise algorithm that transforms the prototype selection problem into a set cover optimization problem and selects prototypes from each class independently. However, the prototypes contain instances of other classes and the radius of the hyperspheres is preselected through the prior experiments. Prototype based learning [48] (PBL) adjusts the radiuses of prototypes by taking into account the classes of instances which a prototype can cover. PBL does not include heterogeneous instances within potential prototypes, as it manages the radiuses of hyperspheres in constructing covering sets.

Figure 6 is the examples of the prototype selection algorithms. The data in this experiment is randomly generated data in this experiment and the total of data is 900. Figure 7(a) is an example of selected prototypes with a fixed radius ( $r = 0.1$ ). A total of 56 prototypes are selected. Figure 7(b) is a hypersphere of variable radiuses. The number of data within the prototype domain is more than one, with a total of 110 prototypes selected. No prior definition of radius is required because the radius is set by considering the different classes of data within the region of each prototype. Figure 7(c) is an example of our hyperrectangle based prototype selection (HRPS) and a number of 47 prototypes are chosen with  $\theta = 0.4$ .

Figure 7 compares IPS, PBL and HRPS in terms of  $\theta$ , data size and time complexity. The test problem consists of three classes of 2D data that are generated at random between 300 and 3000. Figure 8(a) compares the number of prototypes selected as  $\theta$  changes. As  $\theta$  increases from 0.1 to 1.0, the coverage area of a hyperrectangle expands, resulting in fewer

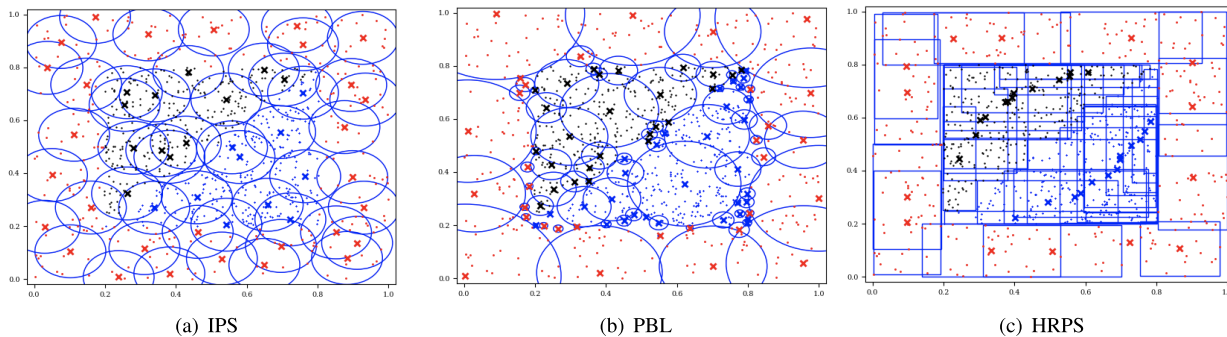


FIGURE 6. Results of IPS, PBL and HRPS for a toy problem.

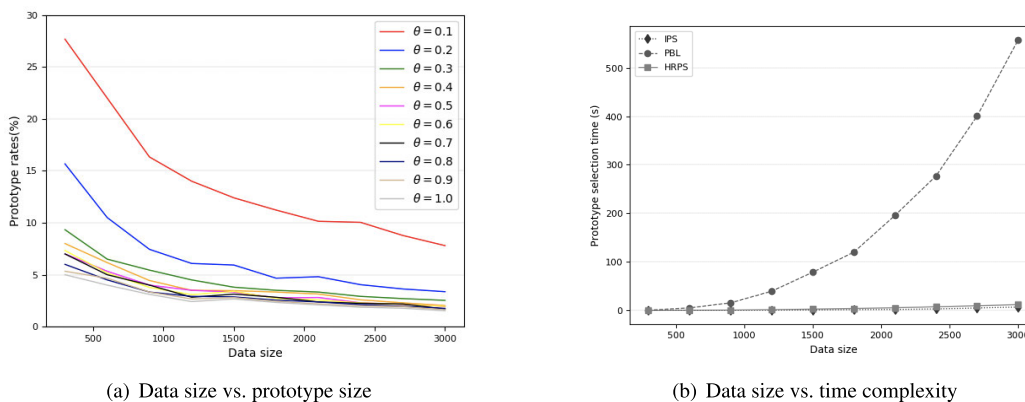


FIGURE 7. Comparison of IPS, PBL and HRPS.

prototypes selected. When a small  $\theta$  is set, a large number of prototypes are selected. So, the maximum number of selected prototypes is equal to the size of the training dataset. A new training dataset, consisting of a small number of prototypes, should be constructed while reflecting the distribution of the input data space. Therefore, it is necessary to find an appropriate  $\theta$  value in order to improve generalization performance.

Figure 8(b) compares the execution time. IPS and HPRS takes lower time than PBL. The reason is because PBL takes time to find hyperspheres with variable radiuses, requiring much computation time. HRPS has a prototype selection time similar to IPS and the runtime remains nearly constant even though the number of instances increases. In the method of dividing the class input space, the method of extending the coverage area of IPS is simpler than that of HRPS.

#### IV. EXPERIMENT

Model experiments were performed on a computer with Intel Xeon(B) Silver 4120 CPUs and Nvidia GPU. The computer supports 256 GB of main memory and 2 CPUs at 2.20 GHz. The GPU model is Tesla V100 and has 32 GB of memory. The GPU was used to compare the training time of CNN models.

The proposed malware feature is compared to the test results of Decision Tree (DT, [49]), Random Forest

(RF, [50]), XGBoost (XGB, [45]) and CNN [51] algorithms. The final learning model was determined through the 5-fold cross-validation for the BIG 2015 dataset. We adopted DT and RF models from `scikit-learn` [52], XGB from XGBoost [53] and CNN from Keras [54]. The whole experiments were conducted by 50 times per cross-validation and analyzed the average of the mentioned metrics for objective comparison.

#### A. ASSESSMENT METRICS

The chosen metrics of the malware detection system are accuracy, recall and precision, balanced accuracy and F1-score under 5-way cross-validation [55]. The predictive result of each malware family was evaluated and their average was analyzed for the overall performance.  $N$  is the size of the test dataset,  $l = \{1, \dots, c\}$  is the class label,  $N_l$  is the number of instances in each class. Let  $\hat{y}_i$  be the prediction result for the  $i^{\text{th}}$  instance of the test dataset and then  $c = 9$  for the BIG 2015 dataset. Each evaluation metric is defined as follows.

Accuracy measures how correctly a model predicts test instances where the basic unit is a single instance. Each unit is weighted equally to the model accuracy.

$$\text{Accuracy} = \frac{1}{N} \sum_{l=1}^c \sum_{i=1}^N 1(\hat{y}_i = y_i = l)$$

A  $c$ -class classifier has a tendency to focus more on classification learning of majority class data rather than minority class data. This makes it difficult to objectively evaluate when the class data is imbalanced. Balanced accuracy can alleviate this problem and is equal to the sum of the proportion of correctly predicted instances divided by the number of classes. This metric is less sensitive to the majority class and gives high weight to data from minority classes. Therefore, the difference between balanced accuracy and accuracy appears when the test dataset shows an imbalanced distribution over the classes.

$$\text{Balanced accuracy} = \frac{1}{c} \sum_{l=1}^c \frac{1}{N_l} \sum_{i=1}^N 1(\hat{y}_i = y_i = l)$$

When evaluating a  $c$ -class classifier, the precision and recall of class  $l$  is computed from the prediction result.

$$\text{pre}_l = \frac{\sum_{i=1}^N 1(\hat{y}_i = y_i = l)}{\sum_{i=1}^N 1(\hat{y}_i = l)}$$

$$\text{rec}_l = \frac{\sum_{i=1}^N 1(\hat{y}_i = y_i = l)}{N_l}$$

Precision  $\text{pre}_l$  of class  $l$  is the ratio of the number of instances correctly answered by class  $l$  to the number of instances predicted to class  $l$ . Recall  $\text{rec}_l$  of class  $l$  becomes the proportion of the correctly predicted instances to the total number of instances in class  $l$ . Precision indicates the correctly predicted proportion of the predicted class data, while the recall analyzes the correctly predicted proportion of the class data. It is a measure that compares the analysis evaluation of a correctly classified specific class through the same class and another class. When both precision and recall are close to 1, the generalization performance of the training model is highly regarded. The overall precision and recall of a  $c$ -class classifier is calculated as follows.

$$\text{Precision} = \frac{1}{c} \sum_{l=1}^c \text{pre}_l, \quad \text{Recall} = \frac{1}{c} \sum_{l=1}^c \text{rec}_l$$

F1-score is the harmonic mean of all the precision and recall values. F1-score calculates the overall average of precision and recall, since the numerator consists of values in the range [0, 1]. This implies that the influence of the majority class has the same importance as the minority class. The high F1-score indicates that the predictive model has good performance, whereas the low F1-score means that it is a poor model.

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

## B. MODEL COMPARISON FOR MALWARE FAMILY CLASSIFICATION

The DT structure was decided through the preliminary experiments with the subsample dataset of a sampling ratio of 30.0% at random. While deciding the DT structure, the tree depth increased from 5 to 30 by step 2. The node splitting criterion of DT uses Shannon's entropy, the minimum number

of a node's instances is set to 4. The internal nodes were applied to split if their number of instances was more than 10. When splitting nodes, the same number of features is checked.

RF and XGB employed 100 decision trees where each tree structure was the same as DT. Similarly, the number of DTs was chosen by changing the number of DTs from 50 to 200 by adding 10 each through the preliminary experiments. When learning RF, all DTs were trained with only 80% of the selected data. The learning rate of XGB was set to 0.05 and the conventional gradient decision tree was chosen. To avoid overfitting, the sample ratio in decision tree construction was 0.7, implying that XGB randomly selects 70.0% of the training data prior to growing trees.

From the related works, the CNN architecture consists of 7 layers as shown in Figure 8. There are 3 convolution layers, one max pooling layer and 3 layers of the fully connected layer. The 5<sup>th</sup> and 6<sup>th</sup> layers are composed of ReLU (Rectified Linear Unit) nodes, and the nodes of the output layer adopted a softmax activation function. The maximum epoch of training was 100 and the mini-batch size was 256. The input layer of the fully connected neural network is configured to prevent overfitting using a 30.0% dropout strategy.

The shape of the proposed 2D feature representation depends on  $\theta$  and  $L$ . To determine the optimal  $\theta$  and  $L$  values, the grid search method was used for repeated experiments.  $L$  changes the discrete value by increasing by 1 from 1 to 20, and  $\theta$  decreases from 1 to  $10^{-5}$ . For a given  $L$ ,  $\theta$  is set to  $\theta = 2^{-\lceil * \rceil \frac{k}{2}} \times 5^{\lceil \frac{k}{2} \rceil - k}$ ,  $k = 0, 1, \dots, 20$ . As  $L$  increased, the continuous improvement was analyzed, but after  $L \geq 6$ , the performance improvement of all models was insignificant. For each  $L$ , when  $\theta = 1$ , the prediction performance showed less than 50%. After  $\theta = 10^{-5}$ , there was no performance improvement, and all training data were determined as prototypes. Therefore, the  $L$  values of 1, 2, 4, and 6 were chosen for the visualization analysis.

Table 4 and Figure 9 compare the performance of the learning models when  $\theta = 0.01$  and  $L$  changes. As  $L$  changes from 1 to 6, the performance of each model tends to increase. For XGB, RF, and CNN, all the metrics are increasing as  $L$  increases. Furthermore, at  $L = 1$ , these learning algorithms achieve above 96.0% and reach 100.0% at  $L = 6$ . However, in DT, as the level of  $L$  increases, the accuracy index shows a tendency to increase, but the changes in recall, precision, and balanced accuracy were observed. When  $L = 6$ , all models achieve their best performance, which is around 98.5% for all evaluation metrics except the DT model. Overall, RF and XGB outperform DT, however the difference in performance is just approximately 5.0% at most.

In comparison of precision, recall, and F1-score, CNN shows a little higher than RF and XGB. When comparing the generalization performance of CNN with that of other algorithms, it shows about 2% to 5% with DT, and a difference of about  $10^{-2}$  with RF and XGB is analyzed. The difference between CNN, RF, and XGB is negligible at all  $L$  values.

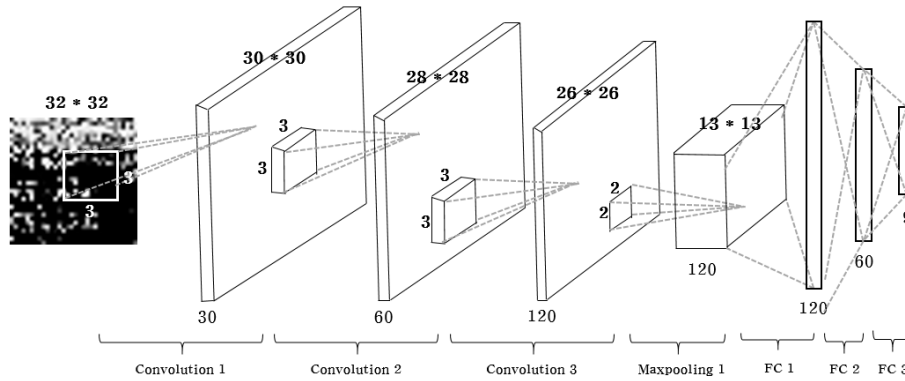


FIGURE 8. CNN structure to detect malware.

TABLE 4. Performance comparison for BIG 2015.

Entropy Histogram Level	Metrics	Model				
		DT	RF	XGB	CNN	GPU
L1	Accuracy	0.9541	0.9794	0.9810	0.9761	0.9785
	Precision	0.9017	0.9596	0.9731	0.9786	0.9579
	Recall	0.9050	0.9620	0.9604	0.9751	0.9501
	F1-score	0.9016	0.9590	0.9652	0.9768	0.9533
	Balanced accuracy	0.9050	0.6200	0.9604	0.9335	0.0504
	Time (sec)	1.2751	4.9507	37.4740	185.2900	21.1384
L2	Accuracy	0.9648	0.9825	0.9862	0.9779	0.9893
	Precision	0.9295	0.9668	0.9778	0.9811	0.9704
	Recall	0.9318	0.9666	0.9708	0.9767	0.9606
	F1-score	0.9303	0.9663	0.9738	0.9789	0.9641
	Balanced accuracy	0.9318	0.9666	0.9708	0.9260	0.0416
	Time (sec)	1.4132	4.0641	57.2040	420.7200	33.3601
L4	Accuracy	0.9675	0.9874	0.9905	0.9851	0.9856
	Precision	0.9160	0.9750	0.9873	0.9861	0.9827
	Recall	0.9409	0.9722	0.9681	0.9849	0.9582
	F1-score	0.9261	0.9730	0.9764	0.9855	0.9670
	Balanced accuracy	0.9409	0.9722	0.9681	0.9629	0.0422
	Time (sec)	1.6416	3.8776	94.1150	1034.8000	58.1106
L6	Accuracy	0.9703	0.9876	0.9925	0.9875	0.9852
	Precision	0.9323	0.9824	0.9882	0.9889	0.9754
	Recall	0.9224	0.9706	0.9711	0.9870	0.9562
	F1-score	0.9262	0.9755	0.9787	0.9880	0.9626
	Balanced accuracy	0.9224	0.9706	0.9711	0.9676	0.0442
	Time (sec)	2.0484	4.1029	133.5200	1658.2000	90.3154

On the other hand, CNN requires significantly more time to train than the other algorithms (Figure 9 (f)). The training time of XGB places 2<sup>nd</sup>. But the training time of CNN exceeds 10 times than that of XGB except the case of  $L = 1$ . DT shows the shortest training time, but RT requires twice as much time as DT.

We evaluated both CPU-based CNN (column CNN in Table 4) and GPU-based CNN (column GPU in Table 4). The two experiments were evaluated similarly, but the training time of the model using the GPU was approximately 9 to 18 times faster. Furthermore, GPU-based CNN was trained faster than XGB in terms of time complexity. As a result, in terms of metrics and training time, GPU-based CNN outperformed XGB.

In the experimental evaluation, the ensemble approach based on the decision tree shows slightly lower performance than CNN, but is analyzed much higher than DT. The gen-

eralization performance of RF, CNN, and XGB from the evaluated scales show high robustness. We found that RF and XGB are more effective at malware classification using the 2D histogram entropy because of their low time complexity and high generalization performance.

### C. MODEL COMPARISON FOR MALWARE DETECTION

The binary classification for malware detection was conducted on both 2-class BIG and 2-class Malwares datasets. The 2-class BIG consists of about 85,000 malware and benign collected in Table 3. In BIG 2015, the Malware Challenge dataset does not include benign examples, so we include the benign dataset of the Malwares dataset to define the classification problem (2-class Malwares) which consists of 30,868 malware and benign.

The Prototype selection rate from 2-class BIG was about 70% (58,878) and analyzed as  $\theta = 0.01$ . This  $\theta$

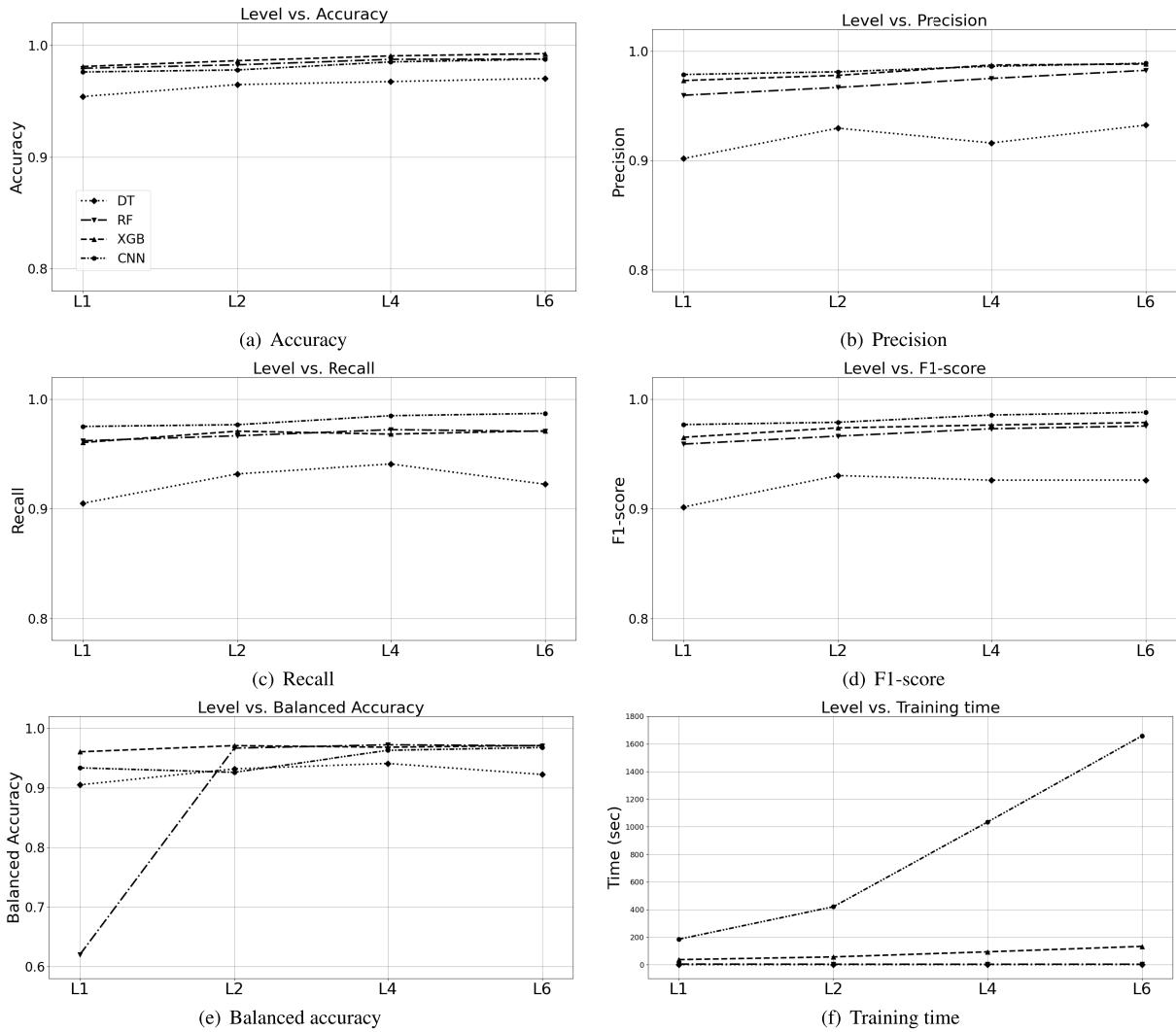


FIGURE 9. Performance comparison.

value was the same value found in the Malware family test, and the addition of benign data did not affect the optimal  $\theta$  value. In the 2-class Malwares dataset, when  $\theta = 0.0001$ , about 49.11% (14,816) prototypes were selected.

Table 5 compares the experimental results of 2-class BIG and 2-class Malwares. The result of 2-class BIG is  $L = 6$ , and 2-class Malwares is  $L = 4$ . From the 2-class BIG results, the performance of all models including DT was analyzed to be higher than 99%. In particular, CNN using GPU approaches 100% in all performance indicators, but requires about 10 times more training time. For 2-class Malwares, the precision, recall, and F1-score of DT do not reach 90%, but the ensemble model approaches 95%. CNN’s precision is about 90%, but recall is 88%. CNNs required several times the training time due to the huge amount of training data. In both experiments, XGB shows higher performance than other models, and is analyzed as a more robust model for malware detection problems.

Based on various thresholds, the precision-recall (PR) curve diagnoses the impact of precision and recall rates on malware classes, whereas the ROC examines the trade-off between false positive and true positive rates in terms of malware and benign instances within the test dataset. Figure 10 shows PR AUC and ROC graphs for 2-class BIG and 2-class Malware. The results prove that the proposed method is effective for malware detection analysis because it does not cause overfitting and the effect of class imbalance. For both the 2-class problems, DT showed the lowest AUC, but showed the highest performance in the order of XGB, RF, and CNN. This trend was similar to malware family detection. The PR AUC and ROC AUC of 2-class BIG are close to 1.0 for XGB, RF, and CNN. The PR AUC of the two types of malware was analyzed to be excellent in the order of XGB, RF, and CNN, and the ROC graph shows the same trend.

**D. ANALYSIS OF THE EFFECT OF PROTOTYPE SELECTION**

A new dataset generated by selecting a prototype is analyzed for its suitability through the case of RF. The model

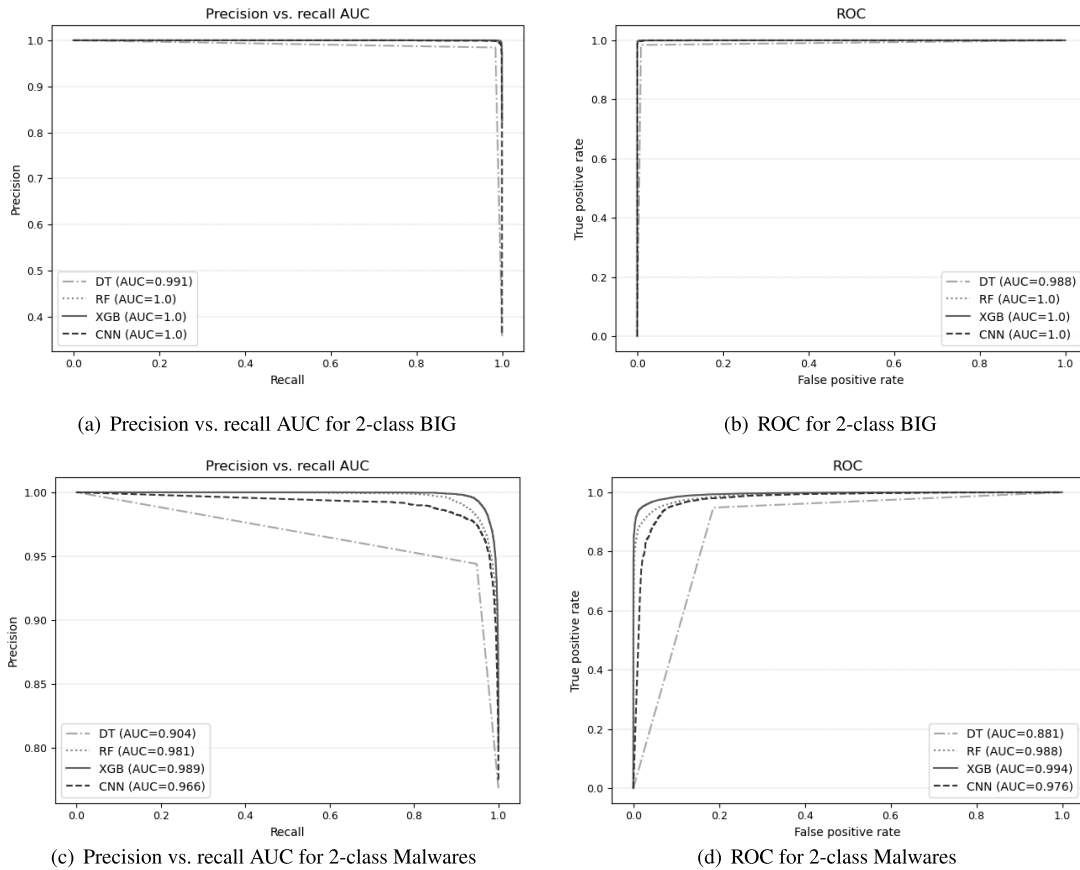


FIGURE 10. Precision-recall AUC and ROC analysis for binary problems.

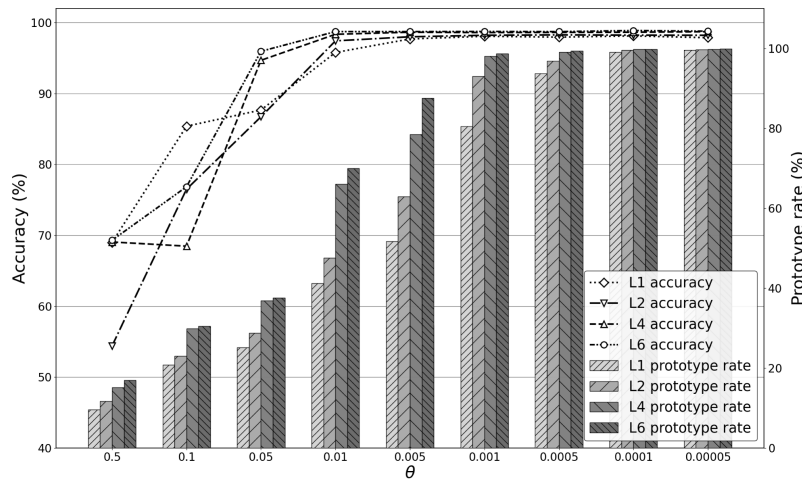


FIGURE 11. Performance of Random Forest by  $\theta$  versus  $L$ .

parameters were the same as in Subsection IV-B. The size of the new training dataset is influenced by the number of prototypes, which is determined by parameter  $\theta$ . Therefore, we compare and analyze between the size of new datasets and RF performance as  $\theta$  and  $L$  change. This type of evaluation

can compare the relationship between the size of prototypes and learning model.

Without loss of generality, a new dataset was generated by scaling the training data to the range  $[0, 1]$  and decreasing  $\theta$  from 0.5 to 0.00005. We compared prototype selection

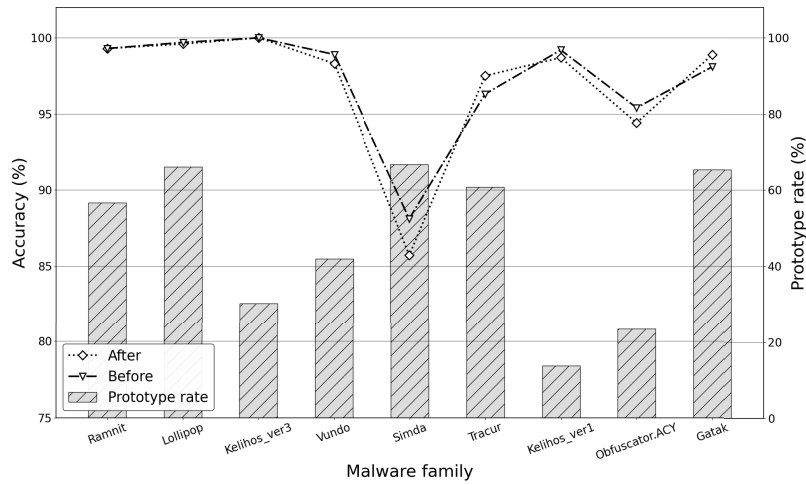


FIGURE 12. Performance comparison of BIG 2015 with Random Forest.

TABLE 5. Performance comparison for malware detection.

Problem	Metrics	Model			
		DT	RF	XGB	CNN
2-class BIG ( $L = 4$ )	Accuracy	0.9904	0.9966	0.9983	0.9956
	Precision	0.9893	0.9965	0.9982	0.9969
	Recall	0.9897	0.9960	0.9981	0.9963
	F1-score	0.9895	0.9962	0.9981	0.9965
	PR AUC	0.9910	1.0000	1.0000	1.0000
	ROC	0.9879	1.0000	1.0000	1.0000
	Balanced error	0.0103	0.0040	0.0019	0.0047
	Time (sec)	18.5505	24.1504	32.6410	198.1232
2-class Malwares ( $L = 6$ )	Accuracy	0.9187	0.9528	0.9668	0.9505
	Precision	0.8878	0.9471	0.9575	0.9022
	Recall	0.8843	0.9191	0.9490	0.8819
	F1-score	0.8860	0.9320	0.9531	0.8839
	PR AUC	0.9041	0.9812	0.9893	0.9662
	ROC	0.8812	0.9883	0.9942	0.9761
	Balanced error	0.1157	0.0809	0.0510	0.0731
	Time (sec)	97.0086	96.8752	363.3640	832.0100

ratios and the accuracy of RF according to the change of  $L$  and  $\theta$  (Figure 10). Because the volume of hyperrectangles decreases as  $\theta$  decreases, the number of selected prototypes approaches closer to the size of the original dataset. In the case of  $L = 6$ , when  $\theta$  changes from 0.5 to 0.01, the accuracy increases in proportion as the number of selected prototypes increases. The number of prototypes increases from 0.005 to 0.00005, but the effect on accuracy is insignificant. The best case occurs at  $L = 6$  and  $\theta = 0.01$  when considering the number of selected prototypes and generalization performance.

Figure 12 compares the performance of the malware family detection by RF when  $\theta = 0.01$  and  $L = 6$ . The average prototype selection rate for each malware family is less than 40.0%. However, the detection rate of Simda is 87.5%, but the detection rate of other families exceeds 95.0%.

The prototype selection rate for Ramnit, Lollipop, Tracur and Gatak is 56.59 % to 66.06 %. The result shows similar or higher performance than before the pro-

toty selection algorithm was applied. We note that the prototype instance representing the class similarly reflects the original class data distribution. In addition, it is expected that the boundaries between malware families are distinguishable to some extent. Kelihos\_ver3, Vundo, Kelihos\_ver1 and Obfuscator.ACY show relatively low prototype selection rates ranging from 13.6% to 41.9%, and similar classification performance. It is anticipated that the instances of these malware families are clustered together and that several family groups are dispersed throughout the feature space.

The performance of Simda is 2.4% lower than that of the others but the prototype selection rate is around 93.0%. The number of Simda instances in the original dataset is too small (0.4%) to reflect the data distribution only with gathered instances. The selected prototypes do not contain sufficient information on its malware family distribution. The same analysis can be considered for Ramnit, Lollipop, Tracur, and Gatak.

## V. CONCLUSION

As malware variants increase, both the time and model complexities are raised for malware classification. To address these challenges, this paper proposed an integrated system of both the fixed size feature design and the prototype selection method based on hyperrectangles. Unlike the previously studied high-dimensional malware features, the histogram entropy benefits from low dimensions, reducing learning time and avoiding overfitting. The hyperrectangle based prototype selection method generates a smaller dataset with more meaningful instances from the original dataset. As a result, the approach can save storage space and training time while retaining high generalization performance. The histogram entropy can be interpretable through 2D images, allowing features to be visually analyzed for comparative analysis. We found that the distinct patterns appeared across cases of the same malware family or that the histogram change trend was consistent.

The proposed approach was shown to be effective in malware analysis using both the BIG 2015 dataset and the Malwares dataset. The performance of the histogram entropy feature was compared using decision tree, ensemble model, and convolution neural network. The histogram features entropy showed high classification performance of approximately 98% or higher, and the ensemble model was the most efficient in terms of learning time and performance metrics. Furthermore, the proposed prototype selection algorithm performed similarly to the entire dataset in malware family classification and malware detection. The overall prototype selection rate was about 50% which could reduce the training time. The proposed feature transformation of malware can account for structural variations per malware family such as entropy peak count, location, shape, etc. There is a need for more research on malware feature design with low dimensionality and prototype selection methods using class boundary instances.

## REFERENCES

- [1] C. LeDoux and A. Lakhota, "Malware and machine learning," in *Intelligent Methods for Cyber Warfare*. Springer, 2015, pp. 1–42.
- [2] A. Sourì and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," *Hum.-Centric Comput. Inf. Sci.*, vol. 8, no. 1, pp. 1–22, Dec. 2018.
- [3] J. Singh and J. Singh, "A survey on machine learning-based malware detection in executable files," *J. Syst. Archit.*, vol. 112, Jan. 2021, Art. no. 101861.
- [4] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on opcode patterns," *Secur. Informat.*, vol. 1, no. 1, pp. 1–22, Dec. 2012.
- [5] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Inf. Sci.*, vol. 231, pp. 64–82, May 2013.
- [6] X. Hu, T. C. Chiueh, and K. Shin, "Large-scale malware indexing using function-call graphs," in *Proc. CCS*, New York, NY, USA, 2009, pp. 611–620, doi: 10.1145/1653662.1653736.
- [7] Y. Ki, E. Kim, and H. K. Kim, "A novel approach to detect malware based on API call sequence analysis," *Int. J. Distrib. Sensor Netw.*, vol. 11, no. 6, Jun. 2015, Art. no. 659101.
- [8] K. Griffin, S. Schneider, X. Hu, and T.-C. Chiueh, "Automatic generation of string signatures for malware detection," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. Berlin, Germany: Springer, 2009, pp. 101–120.
- [9] R. Lyda and J. Hamrock, "Using entropy analysis to find encrypted and packed malware," *IEEE Security Privacy*, vol. 5, no. 2, pp. 40–45, Mar./Apr. 2007.
- [10] N. Nissim, R. Moskovitch, L. Rokach, and Y. Elovici, "Novel active learning methods for enhanced PC malware detection in windows OS," *Expert Syst. Appl.*, vol. 41, no. 13, pp. 5843–5857, 2014.
- [11] S. M. Tabish, M. Z. Shafiq, and M. Farooq, "Malware detection using statistical analysis of byte-level file content," in *Proc. ACM SIGKDD Workshop CyberSecur. Intell. Inform.*, 2009, pp. 23–31.
- [12] N. Bhatia and F. Vandana, "Survey of nearest neighbor techniques," 2010, *arXiv:1007.0085*.
- [13] I. Triguero, J. Derrac, S. Garcia, and F. Herrera, "A taxonomy and experimental study on prototype generation for nearest neighbor classification," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 1, pp. 86–100, Jan. 2012.
- [14] S. Garcia, J. Derrac, J. R. Cano, and F. Herrera, "Prototype selection for nearest neighbor classification: Taxonomy and empirical study," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 3, pp. 417–435, Mar. 2012.
- [15] J. Bien and R. Tibshirani, "Prototype selection for interpretable classification," *Ann. Appl. Statist.*, vol. 5, no. 4, pp. 2403–2424, Dec. 2011.
- [16] R. M. O. Cruz, R. Sabourin, and G. D. C. Cavalcanti, "Analyzing different prototype selection techniques for dynamic classifier and ensemble selection," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 3959–3966.
- [17] J. A. Olvera-López, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, and J. Kittler, "A review of instance selection methods," *Artif. Intell. Rev.*, vol. 34, no. 2, pp. 133–143, 2010.
- [18] S.-S. Choi, S.-H. Cha, and C. C. Tappert, "A survey of binary similarity and distance measures," *J. Syst., Inform.*, vol. 8, no. 1, pp. 43–48, 2010.
- [19] D. Marchette, "Class cover catch digraphs," *Wiley Interdiscipl. Rev., Comput. Statist.*, vol. 2, no. 2, pp. 171–177, 2010.
- [20] R. Younsi and A. Bagnall, "A randomized sphere cover classifier," in *Proc. Int. Conf. Intell. Data Eng. Autom. Learn.* Berlin, Germany: Springer, 2010, pp. 234–241.
- [21] J. Hamidzadeh, R. Monsefi, and H. Sadoghi Yazdi, "IRAH: Instance reduction algorithm using hyperrectangle clustering," *Pattern Recognit.*, vol. 48, no. 5, pp. 1878–1889, May 2015.
- [22] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. 8th Int. Symp. Vis. Cyber Secur.*, 2011, pp. 1–7.
- [23] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proc. 10th Int. Conf. Malicious Unwanted Softw. (MALWARE)*, 2015, pp. 11–20.
- [24] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *Proc. 6th ACM Conf. Data Appl. Secur. Privacy*, Mar. 2016, pp. 183–194.
- [25] D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Using convolutional neural networks for classification of malware represented as images," *J. Comput. Virol. Hacking Techn.*, vol. 15, no. 1, pp. 15–28, Mar. 2019.
- [26] A. Dey, S. Bhattacharya, and N. Chaki, "Byte label malware classification using image entropy," in *Proc. Adv. Comput. Syst. Secur.* Singapore: Springer, 2019, pp. 17–29.
- [27] S. Euh, H. Lee, D. Kim, and D. Hwang, "Comparative analysis of low-dimensional features and tree-based ensembles for malware detection systems," *IEEE Access*, vol. 8, pp. 76796–76808, 2020.
- [28] S. Ni, Q. Qian, and R. Zhang, "Malware identification using visualization images and deep learning," *Comput. Secur.*, vol. 77, pp. 871–885, Aug. 2018.
- [29] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," 2018, *arXiv:1802.10135*.
- [30] A. Nappa, M. Z. Rafique, and J. Caballero, "The MALICIA dataset: Identification and analysis of drive-by download operations," *Int. J. Inf. Secur.*, vol. 14, no. 1, pp. 15–33, Feb. 2015.
- [31] *Virus Total*. Accessed: Feb. 21, 2021. [Online]. Available: <https://www.virustotal.com/gui/>
- [32] *Vx Heaven*. Accessed: Feb. 21, 2021. [Online]. Available: <http://vxheaven.org/>
- [33] *VIPRE*. Accessed: Feb. 21, 2021. [Online]. Available: <https://www.vipre.com/>
- [34] *Analyzing Unknown Binaries*. Accessed: Feb. 21, 2021. [Online]. Available: <http://anubis.isecslab.org/>
- [35] *Malwares*. Accessed: Feb. 21, 2021. [Online]. Available: <https://www.malwares.com>
- [36] P. Burnap, R. French, F. Turner, and K. Jones, "Malware classification using self organising feature maps and machine activity data," *Comput. Secur.*, vol. 73, pp. 399–410, Mar. 2018.



- [37] Y. Fan, Y. Ye, and L. Chen, "Malicious sequential pattern mining for automatic malware detection," *Expert Syst. Appl.*, vol. 52, pp. 16–25, Jun. 2016.
- [38] D. Yuxin and Z. Siyi, "Malware detection based on deep learning algorithm," *Neural Comput. Appl.*, vol. 31, no. 2, pp. 461–472, Feb. 2019, doi: [10.1007/s00521-017-3077-6](https://doi.org/10.1007/s00521-017-3077-6).
- [39] K. S. Han, J. H. Lim, B. Kang, and E. G. Im, "Malware analysis using visualized images and entropy graphs," *Int. J. Inf. Secur.*, vol. 14, no. 1, pp. 1–14, Feb. 2015.
- [40] J. Luo and D. C. Lo, "Binary malware image classification using machine learning with local binary pattern," in *Proc. IEEE Int. Conf. Big Data*, Boston, MA, USA, Dec. 2017, pp. 4664–4667.
- [41] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *J. Comput. Secur.*, vol. 19, no. 4, pp. 639–668, 2011.
- [42] X. Hu, K. G. Shin, S. Bhatkar, and K. Griffin, "MutantX-S: Scalable malware clustering based on static features," in *Proc. Annu. Tech. Conf.*, 2013, pp. 187–198.
- [43] *Cuckoo*. Accessed: Feb. 21, 2021. [Online]. Available: <https://cuckoosandbox.org/>
- [44] I. Sorokin, "Comparing files using structural entropy," *J. Comput. Virol.*, vol. 7, no. 4, pp. 259–265, Nov. 2011.
- [45] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," 2016, *arXiv:1603.02754*.
- [46] V. V. Strelkov, "A new similarity measure for histogram comparison and its application in time series analysis," *Pattern Recognit. Lett.*, vol. 29, no. 13, pp. 1768–1774, Oct. 2008.
- [47] V. V. Vazirani, *Approximation Algorithms*, vol. 1. Berlin, Germany: Springer, 2001.
- [48] D. Hwang and Y. Son, "Prototype-based classification and error analysis under bootstrapping strategy," *Int. J. Data Mining, Model. Manage.*, vol. 10, no. 4, p. 293, 2018.
- [49] J. R. Quinlan, "Learning decision tree classifiers," *ACM Comput. Surv.*, vol. 28, no. 1, pp. 71–72, Mar. 1996, doi: [10.1145/234313.234346](https://doi.org/10.1145/234313.234346).
- [50] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 8, pp. 832–844, Aug. 1998.
- [51] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: <http://www.deeplearningbook.org>
- [52] *Scikit-Learn*. Accessed: Feb. 21, 2021. [Online]. Available: <https://scikit-learn.org/>
- [53] *XGBoost*. Accessed: Feb. 21, 2021. [Online]. Available: <https://xgboost.readthedocs.io/en/latest/index.html>
- [54] *Keras*. Accessed: Feb. 21, 2021. [Online]. Available: <https://keras.io/>
- [55] J. Kelleher, B. Mac Namee, and A. D'Arcy, *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies*, 2nd ed. Cambridge, MA, USA: MIT Press, 2020.



**BYUNGHYUN BAEK** received the B.S. and M.S. degrees from Dankook University, South Korea. He is currently a Researcher at the Digital Healthcare Institute, GI VITA, South Korea. His research interests include machine learning, parallel processing, and image processing.



**SEOUNGYUL EUH** received the B.S. and M.S. degrees from Ajou University, South Korea. He is currently pursuing the Ph.D. degree with the Department of Software Science, Dankook University, South Korea. He is also the Vice President with the Security Technology Institute, KSign, South Korea. His research interests include machine learning, parallel processing, and threat intelligence.



**DONGHEON BAEK** received the Ph.D. degree from Seoul National University, South Korea. He was a Public Health Doctor at the Korean Food & Drug Administration and worked on regulatory affairs of medical devices certification. He is currently a Professor with the Department of Oral Microbiology and Immunology, School of Dentistry, Dankook University, South Korea. His research interests include the T-cell immunity, pathogenicity of periodontal diseases and evaluation of public healthcare devices, bioinformatics and applications, and information systems.



**DONGHOON KIM** received the M.Sc. degree from Auburn University, USA, and the Ph.D. degree from North Carolina State University, USA. He is currently an Associate Professor with the Department of Computer Science, Arkansas State University, USA. He formerly worked at Samsung Electronics, South Korea, as a Software Engineer. His research interests include cybersecurity, machine learning, high-performance computing, and software engineering.



**DOOSUNG HWANG** received the Ph.D. degree from Wayne State University, USA. He is currently a Professor with the Department of Software Science, Dankook University, South Korea. Previously, he was a Senior Researcher at the Electronics and Telecommunications Research Institute (ETRI), South Korea, and worked on learning algorithm design and intelligent systems, such as expert systems, image recognition, time-series analysis, and parallel computing.

...