Spring 2018

Image Robust Hashing for Malware Detection

Wei-Chung Huang
San Jose State University

Image Robust Hashing for Malware Detection

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Wei-Chung Huang

May 2018

The Designated Project Committee Approves the Project Titled

Image Robust Hashing for Malware Detection

by

Wei-Chung Huang

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2018

Mark Stamp	Department of Computer Science
Katerina Potika	Department of Computer Science
Fabio Di Troia	Department of Computer Science

ABSTRACT

Image Robust Hashing for Malware Detection

by Wei-Chung Huang

This research is focused on a novel approach to detect malware based on static analysis of executable files. Specifically, we treat each executable file as a two-dimensional image and use robust hashing techniques to identify whether a given executable belongs to a particular family or not. The hashing stage comprises two steps, namely, feature extraction, and compression. We compare our robust hashing approach to other machine learning-based techniques.

ACKNOWLEDGMENTS

I would like to thank my family for their support. Furthermore, I would like to express my deep gratitude to my research supervisor, Dr. Mark Stamp, for his patient guidance.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Malware is software that is intentionally designed to cause harm to computer systems [1]. Due to our heavy reliance on computers in general, and software in particular, malware detection is a vitally important topic in information security. Signature scanning (i.e., pattern matching) is the most common form of malware detection, and hence malware writers have developed many concealment strategies aimed at defeating standard signature scanning techniques [1]. These concealment strategies result in malware families consisting of large numbers of related malware variants. In such cases, malware detectors must focus on strategies aimed at entire families, rather than individual malware samples.

Malware detection can be based on static analysis or dynamic analysis, or a combination of the two. As the names suggest, static malware detection is based on features that can be extracted without executing (or emulating) the code. For example, mnemonic opcodes are an example of a static feature [2, 3]. On the other hand, dynamic malware detection inspects the behavior of software, that is, the software is executed (or emulated) and features are extracted. In any case, the resulting features are then used to classify samples as malware or benign. In this research, we rely on static features and we apply techniques from image robust hashing [4] and machine learning to analyze these features

The remainder of this paper is organized as follows. In Chapter 2 we discuss relevant background topics, including the malware data format in Section 2.2, the introduction of robust hashing in Section 2.3 and Section 2.4, the features and machine

learning techniques that we use will be discussed in Section 2.5 and Section 2.6. In Chapter 3 we discuss our dataset in Section 3.1 and show the experiment result in Section 3.2 and Section 3.3. Then Section 3.4 gives the comparison of the result from using robust hashing approach to other machine learning-based techniques. In the end, we give the conclusion and discuss the future work in Chapter 4.

CHAPTER 2

Background

2.1 Previous Work

In [5] they visualized raw binary data in executable files such as data files and process memory as image, in [6] they proposed a method to visualize and classify malware using image processing techniques where they are mainly used to extract features to be classified. In [6] they concluded that the variation caused by concealment strategies would be ignored if we view them as an image. In [7], Tian et al suggest that the function length plays a significant role in classifying Trojans and they achieved 88% average accuracy over 7 different types of Trojans and 721 malware samples. In contrast to [7], [6] uses GIST [8] feature and the classifier, k-nearest neighbors, to obtain average classification accuracy of 98% on 9,458 samples with 25 malware families.

In [4] they proposed a robust image hashing to handle the proliferation of digital images where it can be used in many applications such as managing huge image databases, image indexing, or image authentication [9, 10, 11]. Especially for the hashing method of [11], it uses both global and local features where global features are based on luminance and chrominance, and local features are based on the image textures whereas in [12] they discussed the mathematical framework for the studies of texture perception.

The scheme of feature extraction in the robust image hashing was proposed by [13] where the research partitioned the process of retrieving an image hash into two steps, feature vector extraction and compression. The extracted feature in [13]

was also proposed by the same author of earlier research in [14]. And [15] showed an evidence of distinguishing malicious manipulations from JPEG compression which is based on the Discrete Cosine Transform (DCT) coefficient. The methods of [4] include an image feature extraction by using a wavelet decomposition, a quantization, an error-correcting, and a hash value measurement which is called equality percentage (EP), the percentage of equal vector components. [4] also achieved close to 100% EP for resistance to attacks in 100 images and achieved 45% to 65% for the test of collision with unrelated images.

Another novel robust image hashing was found in [16], which proposed a dither-based secure image hashing using distributed coding. The research had adopted the idea of distributed source coding with Wyner-Ziv encoder [17] and a dithering process which can be viewed as a side information in the distributed coding scheme and as a way to amortize a quantization error for the similar input. Based on Wyner-Ziv encoder, [16] proposed to treat the syndrome as the final hash value. In the experiment result, [16] tested for 3 images in 12 variation attacks and got 16% difference in Hamming distance manner. Moreover, the research tested 15 pairs of distinct images on a collision experiment then achieved about 40% difference in hamming distance manner.

2.2 Malware Images

As [6] proposed, Nataraj et al transformed a malware executable file into a two-dimensional image. Basically, the transformation process takes the raw byte data in an executable file as a pixel value, and then concatenate every byte (gray-level pixel) to form a two-dimensional image. Figure 1 shows the process how it works. It is worth to note that when generating a two-dimensional image it takes width and

height as a factor but here it only has the overall image length, which is the size of width plus the size of height.

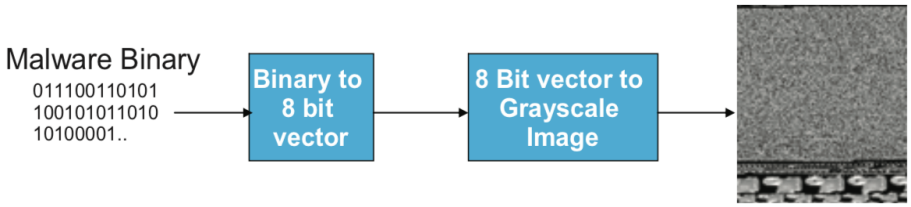

Figure 1: Visualizing malware as an image.

The malware obfuscation might be mitigated when viewing it as an image. Some malware pictures that are shown in Figure 2 indicate the similarity between different variations of the same family. Figure 3 includes more examples for different families. If we look closely at the transformed executable files in Figure 4, it is clear to tell that there are some fragments showing the structure of the executable file. For example, .rdata section represents read-only data such as literal strings and debug directory information, and black padding sections show the zero padding. More detail about the binary fragment can be found in [18].

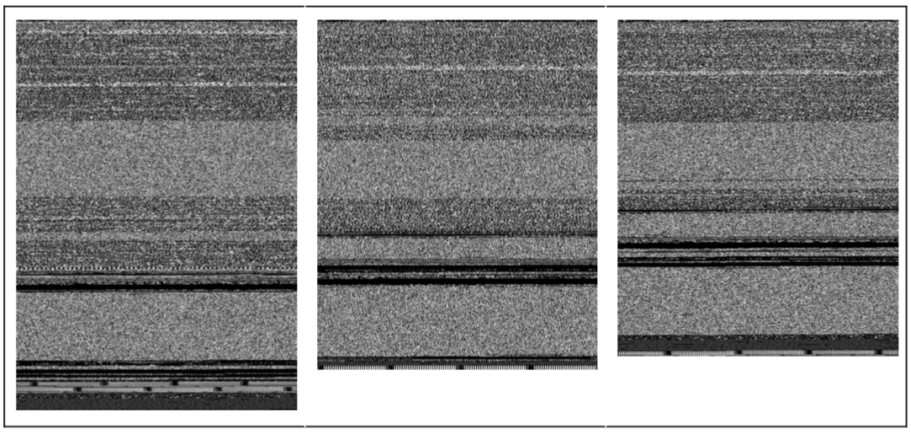

Figure 2: These three malware images are type of Trojan Downloader which is belonging to the same family named Swizzor.gen!E.

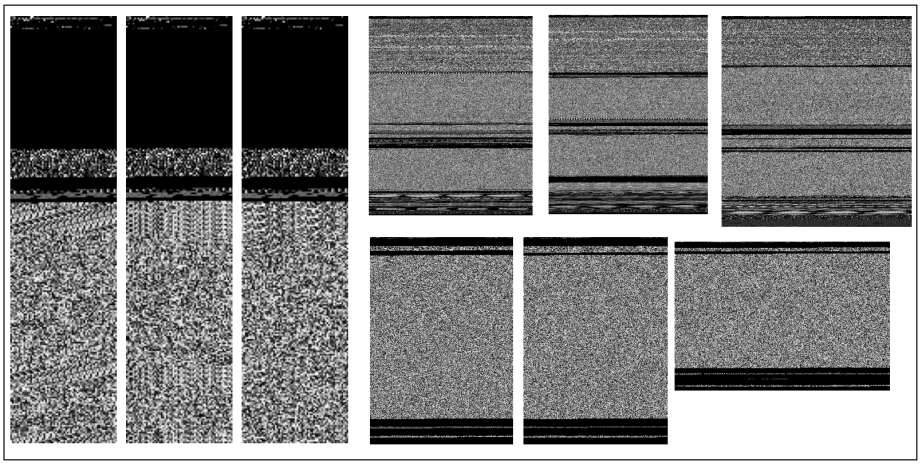

Figure 3: More example of malware images. The left three images are family Agent.FYI. The right top three images are C2LOP.P, and the right bottom three images are Alueron.gen!J

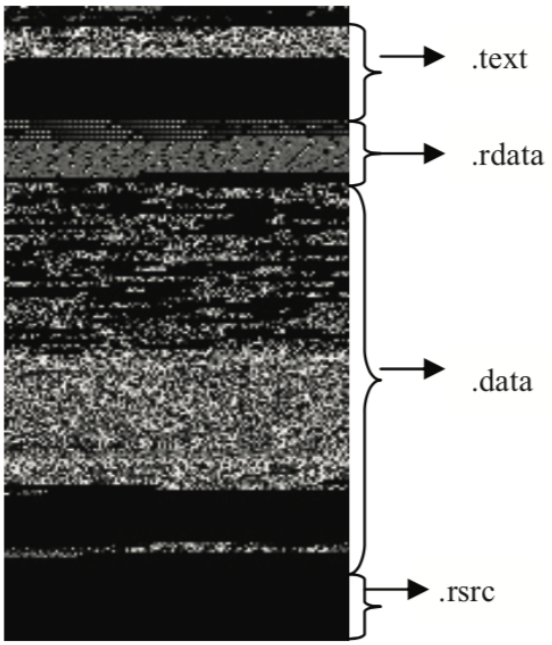

Figure 4: Various sections of an malware image sample.

2.3 Hashing Function

In some sense, hashing means a mapping from something to something. From the perspective of computer science, a hash function is any function that maps the

data to a specific data of fixed size. On the one hand it provides an efficient way to retrieve the data from a given hash value. On the other hand, different data could be mapped to the same hash value, that is, collision. Here we skip normal hash functions, instead, we look into two specific types of hashing, cryptographic hashing, and robust hashing.

2.3.1 Cryptographic Hashing

A cryptographic hash function generally is used in cryptographic applications, such as signature identification [19], and some hashing functions such as MD5 and SHA-1 or even the improved version [20] are pretty popular nowadays. The reason why it is suitable for cryptographic uses due to the following properties have to be satisfied [21].

- Deterministic: No matter what size of input, it will output the small fixed number of bits.
- Efficiency: It has to be efficient to compute any give data.
- One-way: It is infeasible to construct a message from its hash value except for doing an exhaustive search.
- Collision-resistance: It is infeasible to find a collision.
- Sensitivity: A small change of the input message causes the hash value extensively.

2.3.2 Robust Hashing

The significant difference from cryptographic hashing is that robust hashing tends to preserve the change of the message, that is, sensitivity property is traded off

such that it creates more collision. Although it sounds a little counter-intuitive, it can be used to identify similar messages or data object. Clearly, it is a good technology for authentication problems. For example, Biometric Authentication [22], Image Watermark authentication [9], or Image Indexing [4].

As mentioned before, many applications of robust hashing are surrounded by the field of image applications. Usually, images are subject to many modifications such as JPEG compression, image adjustment, or malicious modification. Even those modifications are applied, the modified images still look the same or similar. Obviously, that adjudgment is based on the decision from human, not computer. So robust hashing comes into the picture. Its goal is to identify similar visual look of images by hashing them to the same hash value. And basically, the process of image robust hashing can be seen at Figure 5 proposed by [13]. In Figure 5, the feature extraction step is about to retrieve useful information in the image, and the goal of compression step is, explicitly, to compress the intermediate hash value, or implicitly, to reduce minor difference or noise such that similar hash values can be clustered to the same group.

There are two main approaches proposed in the step of compression, [4] proposed using an error-correction decoding to cancel small perturbations, and [16] proposed using distributed source coding schema comprising a dithering process and a Wyner-Ziv Encoder [17].

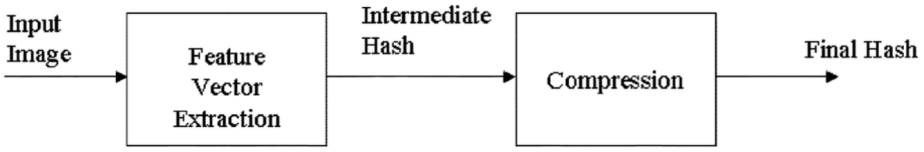

Figure 5: Process of robust hashing.

2.4 Compression

Data compression has been widely used in many fields, for example, audio processing, video processing, or data transmission, cryptographic, etc. Simply speaking, it is about to compress the data by eliminating the less informative portion of data. In terms of digital signal, it reduces bits from the original data bit-stream. Data compression can be categorized into lossless and lossy compression. As the name suggests, lossless compression reduces the size of data by eliminating statistical redundancy, whereas lossy compression aggressively removes "unnecessary" bits. In this research, we are more interested in lossy compression since the difference introduced in the intermediate hash value are expected to be compressed or removed. Furthermore, we choose to use the well-known compression technique based on an image, that is, JPEG compression, since the data format in our research is a two-dimensional image.

2.4.1 JPEG Compression (Encoding)

JPEG stands for Join Photographic Experts Group, is a very common lossy compression method used in 2D digital images. It is a lossy compression based on the frequency domain, that is, the Discrete Cosine Transform (DCT) comes into the picture. Rather than introducing its tedious history, we briefly explain the major steps that are used in the compression.

1. Discrete Cosine Transform (DCT)

   DCT is generally compared with the Discrete Fourier Transform (DFT), where they both aim to decompose a discrete-time vector from the spatial domain to frequency domain with using different basis functions. DFT uses a set of complex exponential functions while DCT uses cosine functions. Despite several variants of the DCT, we choose the most common form, the DCT, or called

DCT-II. We compute a 1D DCT-II as

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \quad k = 0, ..., N-1$$

where the N real numbers x_i are transformed into the N real numbers X_i. Hence, a 2D DCT-II is given by

$$X_{k_1,k2} = \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x_{n_1,n_2} \cos\left[\frac{\pi}{N_2}(n_2+\frac{1}{2})k_2\right] \cos\left[\frac{\pi}{N_1}(n_1+\frac{1}{2})k_1\right]$$

In the normal process of JPEG compression, an 8x8 pixel-based block-wise DCT is applied to produce DCT coefficient matrices which represent the response to particular frequency revealing a frequency distribution. The following shows an example of DCT transformation with the source Figure 6 and a coefficient view applied with 8x8 blocks in Figure 7. As it shows, the overall structure is visually recognizable due to the high frequencies along with the edges. This observation is also proven by the fact that humans are more sensitive to the energy of high frequency than the low frequency in an image. Hence, the next step of the JPEG compression is to reduce the low-frequency part of the image.

2. Quantization

This quantization step takes the most critical and also the unique part of compression. In general, quantization is a process of constraining the data from a large set to a small set. For instance, converting a real number to an integer number is always a good example. However, the metric that is used in quantization really depends on the application. A good metric would result in a good compression (low distortion rate) while a bad metric might just reduce the set size abruptly. Recall that for the JPEG compression, we want to keep informative data (high frequency) and reduce the low-frequency part where humans can rarely distinguish the difference. Since we have a DCT coefficient

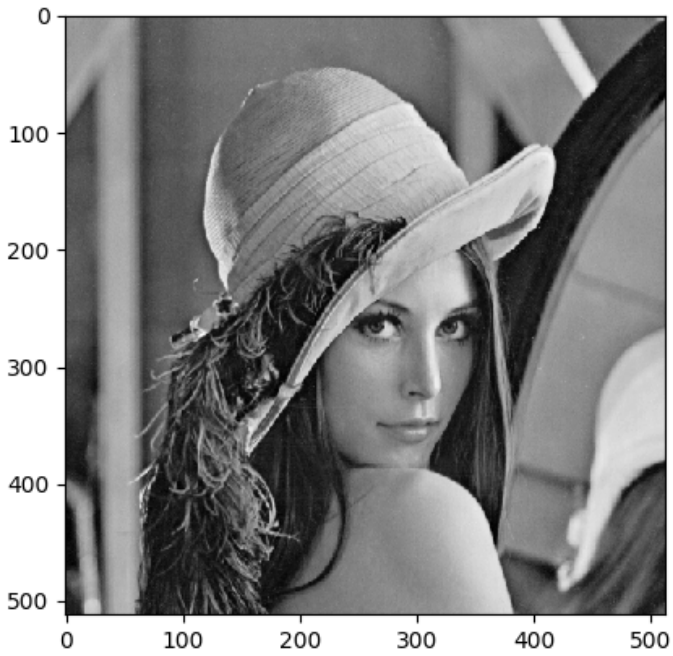

Figure 6: Source image of JPEG compression.

map, it is common to quantize by dividing it with an 8x8 matrix, or so-called Quantization Matrix where having larger matrix values in low frequencies and smaller values in high frequencies. Figure 8 shows an example of quantization by reducing low frequent bits from the DCT coefficient map in Figure 7.

3. Zig-Zag Scan

   Since the low frequent bits are reduced, the DCT coefficient map may contain continuous zero values. As Figure 9 denotes, this step introduces a way to generate a coefficient vector by starting from the high frequency to the low frequency in a zig-zag order. On one hand, a general JPEG compression can use Huffman coding on what is left to reduce the data size. On the other hand, the vector can be served as a frequency distribution of the given image.

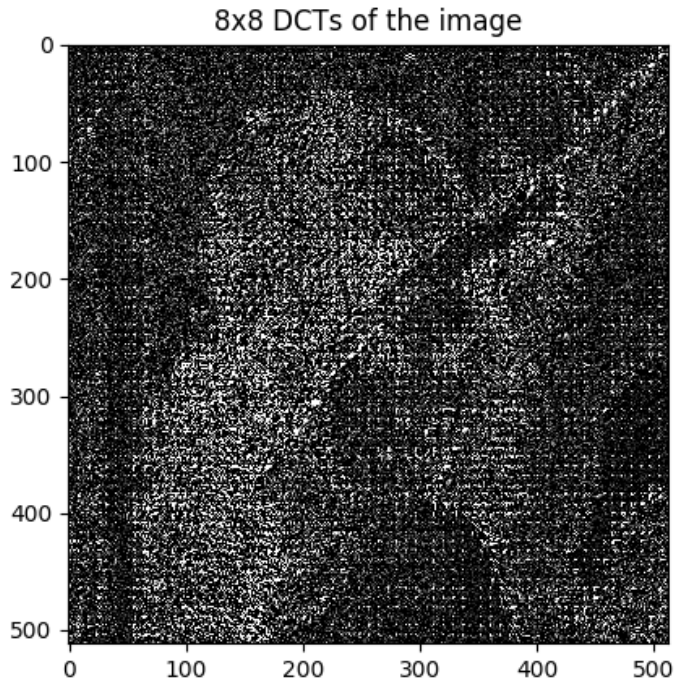

Figure 7: 8x8 DCT coefficients

2.4.2 Channel Coding

Channel coding is a term for the error control which is used in a communication system. In the telecommunication theory, messages are sent as a bit-stream and are subject to channel noise. Thus, some error could be introduced from the sender to the receiver. Error control and then comes into the picture to either detect or correct the error, that is, ensuring the message received at the receiver is likely the same as the original as possible. For the error correction, there are two main ways to deal with by sending extra bits along with the message as a header or "solution". The first one is Automatic Repeat Request (ARQ), where the receiver will request a resend if an error is detected. The second one is Forward Error Correction (FEC), where no resend request is made but the message would be encoded in some way to be able to

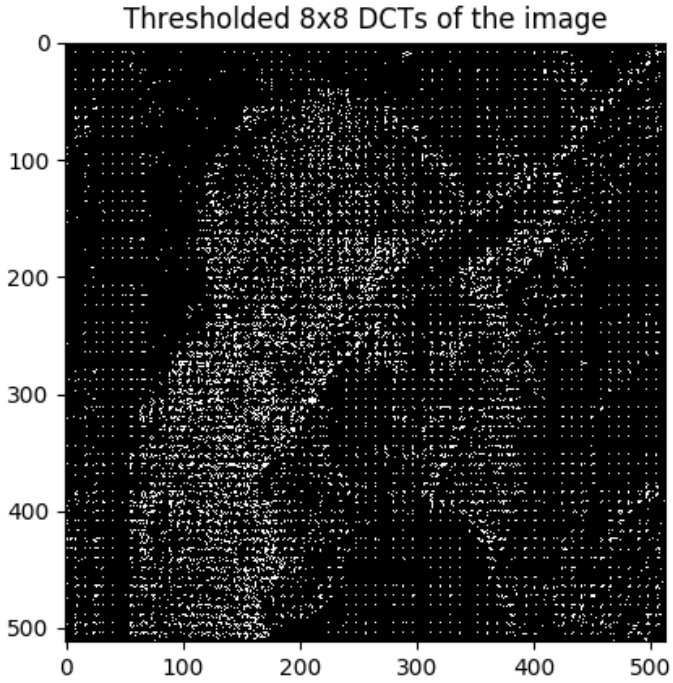

Figure 8: Quantized 8x8 DCT coefficients

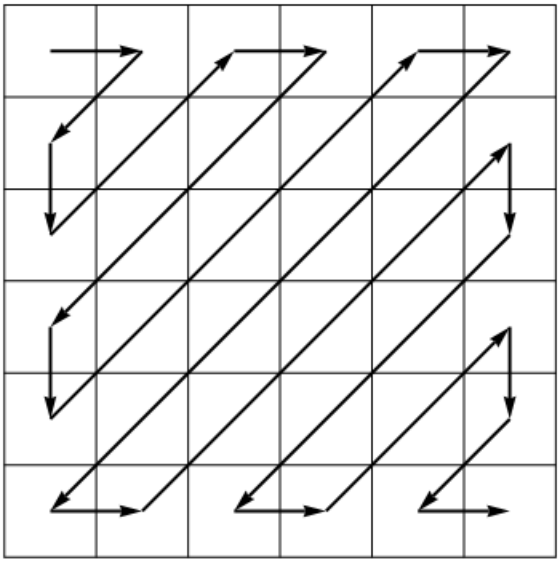

Figure 9: An example of zig-zag scan of the DCT coefficient map.

be recover. In our research, we are more interested in viewing the difference from each image with the same family as the channel noise, and then we adopt a well-known error-correcting method, Hamming codes [23], to eliminate the noise (differences).

2.4.2.1 Hamming Encoding

Hamming codes is a family of linear error-correcting codes. It has the ability to correct the message given to the following conditions. Given an integer $r \geq 2$, there is a codeword with length $n = 2^r - 1$, and message length $k = 2^r - r - 1$. For instance, Hamming(7, 4), which is a popular configuration in the Hamming codes, can correct 1 bit error by carrying 4 bits data with 3 extra parity bits. Figure 10 shows the concept of how it works. Within the 7 bits transmitted message, p_1, p_2, p_3 are the parity bits and d_1, d_2, d_3, d_4 are data bits. As the figure shows, p_1 determines d_1, d_2, d_4 so on and so forth. In other words, d_1 only contributes p_1 and p_2, d_2 only contributes p_1 and p_3. This implies that if d_1 is flipped, then only p_1 and p_2 will be affected. However, the problem would become how to associate each parity bit with data bit such that we can validate p_1 and p_2 with d_1 accordingly. Fortunately, p_1 can also be validated by d_2 and p_3.

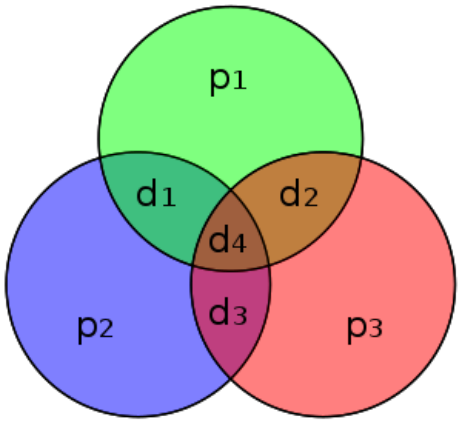

Figure 10: Hamming(7,4) example.

The idea behind is using an even parity check. Given the coverage of the diagram in Figure 10, each parity bit will be determined by satisfying an even count of bits with the followed data bits. For example, Table 1 shows an example of how to produce p_1 by its coverage. If d_1 and d_2 is 0 and d_4 is 1, and then p_1 will be set to 1. These parity bits are also called the syndrome vector, which will be transmitted along with the message.

Table 1: Example of generating parity bit in Hamming(7, 4)

p_1	d_1	d_2	d_4
0	0	0	0
1	0	0	1
1	0	1	0
0	0	1	1
1	1	0	0
0	1	0	1
0	1	1	0
1	1	1	1

2.4.2.2 Hamming Decoding

In the decoding stage, we briefly introduce how to use the syndrome to reconstruct the original message. Follow the discussion above, once the syndrome is produced, we can simply validate or locate the error bit by utilizing its even-count property. Strictly speaking, the received syndrome will be used to check if the even-count property is still held. If not, then we claim that there is an error in the specific coverage. Table 2 shows the un-satisfied condition for each possible transmitted bit is flipped. For example, assume d_1 is flipped, after checking the even-count property, the coverage will give a unique combination to indicate d_1 is flipped. That is, we know d_1 is altered because p_1 and p_2 are not satisfied as expected. As Table 2 sug-

gests, each flipped bit will cause a unique 3-bit combination. Therefore, by using this mechanism, some channel noise can be corrected.

Table 2: A view of error-correcting in Hamming(7,4)

coverage	d_1	d_2	d_3	d_4	p_1	p_2	p_3
$p_1 \rightarrow d_1\ d_2\ d_4$	x	x	o	x	x	o	o
$p_2 \rightarrow d_1\ d_3\ d_4$	x	o	x	x	o	x	o
$p_3 \rightarrow d_2\ d_3\ d_4$	o	x	x	x	o	o	x

2.4.3 Distributed Source Coding

Distributed source coding (DSC) [24] is a concept of reducing the computational burden from an encoder side. But it is subject to multiple correlated information do not communicate with each other. How it works is by putting some information to the decoder as a side information. Take video compression as an example. Assume we have many embedded camera sensors deployed around Central Station to collect pictures for an arbitrary analysis. The simple approach is letting each sensor encodes a picture, sends it back to the server, and then the server will decode it accordingly. However, what if there are some image noise need to be cleared, each sensor will then have to apply a de-noise task before the compression and transmission. This certainly reduces an overall efficiency because the noise each sensor received is similar. As a result, DSC comes into the picture, the noise will be viewed as a side information and acquired directly on the decoder side.

Wyner-Ziv encoder [17] is a famous example where it argues that a Wyner-Ziv coder depends almost only on the source data, not the side information. Also, [16] gives a conceptual view of DSC in Figure 11.

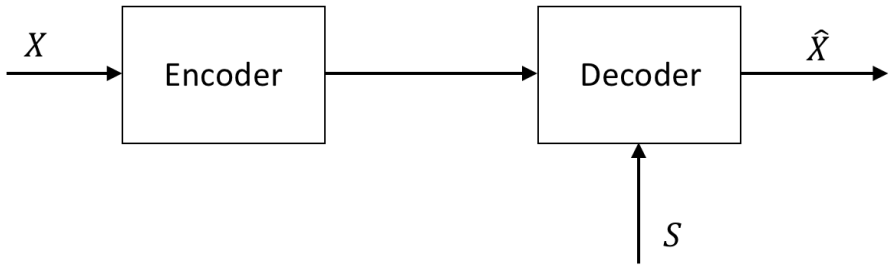

Figure 11: Distributed source compression.

2.5 Image Features

There are many kinds of features in the field of image classification. In general, it can be categorized to local and global features. The local feature represents the modifications or differences in a small region of interest, such as textures. On the other hand, global feature tends to represent the global structure of images, for example, edges. By observing some malware images, we decide to use the following eight features from local and global groups and two of global features are used in robust hashing. It is also worth noting that malware images are all in gray-level view, that is, we only consider one-channel cases.

2.5.1 Local Feature

The local feature is commonly used in texture analysis, texture synthesis. Although there is no strict definition of texture, normally, people will say the repeated patterns can be viewed as texture. Here we use two common local features attempting to extract the repeated patterns of malware image.

2.5.1.1 Local Binary Patterns

In the recent decade, Local Binary Patterns (LBP) is a well-known texture descriptor in facial recognition [25]. The idea of LBP is extracting the local pixel contrast while preserving the information of pixel position. Figure 12 shows the uniform LBP feature extraction process. Basically, the image is separated by blocks, and each block is divided by cells, and each cell computes a histogram with respect to the center of the cell. Finally, combine those local histograms so that we will get an LBP descriptor of a feature vector.

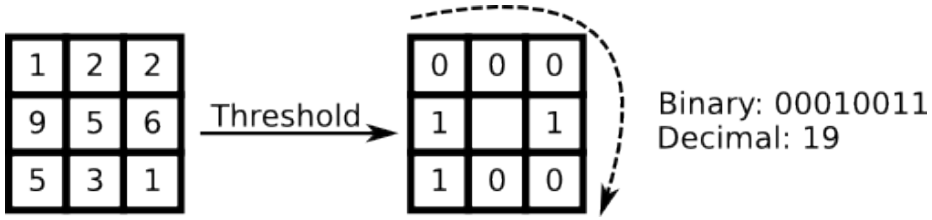

Figure 12: Process of extracting LBP feature.

2.5.1.2 Histogram of Oriented Gradient

Another texture descriptor is Histogram of Oriented Gradient (HOG). Similar to other scale-invariant feature transform descriptors, HOG has been wildly used in object detection because of its robustness on appearance, geometric and photometric. For example, human detection [26]. Unlike LBP using contrast, HOG uses a gradient as its local descriptor. However, they share the process of dividing an image into blocks and cells. Figure 13 shows how orientation histograms produce the feature vector.

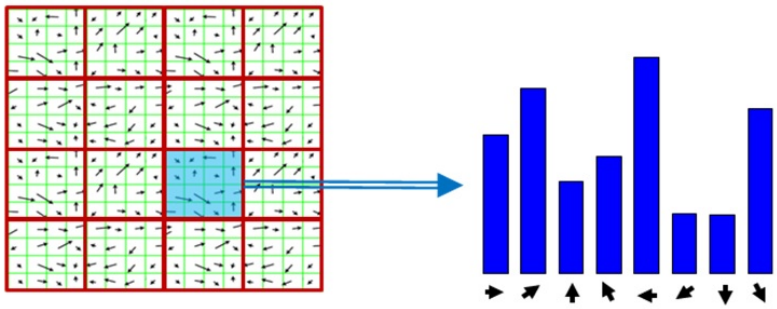

Figure 13: Process of extracting HOG feature.

2.5.2 Global Feature

As the name suggests, the global feature is focusing on the entire image object instead of local regions. In other words, the global feature can reflect the struct of images. We observed that there are many commonplaces with respect to structure such as black lines, lightness, and overall similarity.

2.5.2.1 Horizontal Edge

Edge is a very strong feature that has been widely used in many digital image processing applications. It can also be used in combined with other techniques or be used standalone. In our problem, we focus on those horizontal black lines with regards to its vertical position. That is, we want to extract the distribution of all horizontal edges in the image by projecting the magnitude on the edge map onto a one-dimensional space. Figure 14 shows the flow of feature extraction process. Note that we apply a low-pass filter in the middle process to enhance the edge.

To deal with the variant size of images, the pixel-based edge extraction could be too sensitive to have a good representation. Therefore, instead of pixel-based approach, we project it block-by-block. Figure 15 shows the distribution after the projection. To better show the relationship between the feature and the source image,

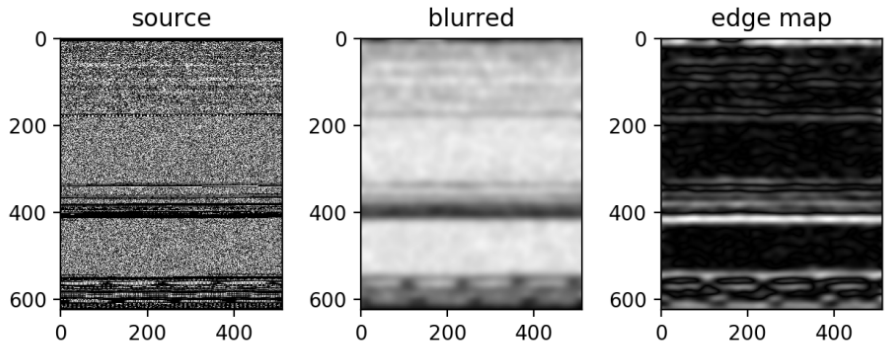

Figure 14: Process of obtaining the horizontal edge map (from left to right).

Figure 16 displays a side-by-side comparison in which the middle figure is the rotated view from Figure 15.

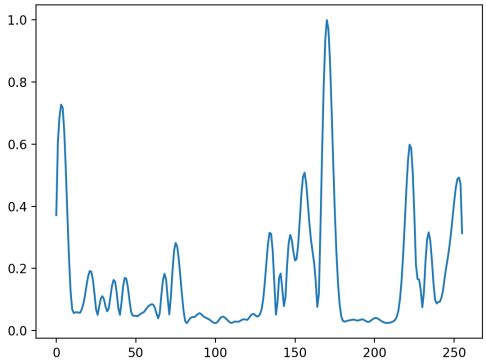

Figure 15: Feature vector of horizontal-edge.

2.5.2.2 Pixel Intensity

The idea of this feature is to find the image-to-image similarity by measuring all pixel intensity values. Due to the variant size of images, we compute the pixel mean value based on grids over an entire image.

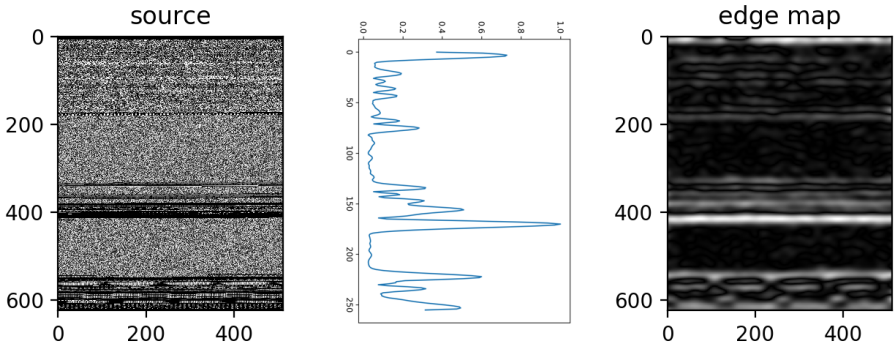

Figure 16: Feature vector of horizontal-edge. Side-by-side comparison with the source image.

2.5.2.3 Contrast

Contrast is one of the features that has been broadly used. Intuitively, contrast tells us the variance for the brightness and darkness with regards to two different image regions. However, it is not straightforward to present the difference with more than two objects in a single value or vector. As a result, histogram comes into the picture, and we generate a histogram for an entire image as a feature vector.

2.5.2.4 Median Filter

A median filter is one of the low-pass filters for which many de-noise techniques are used to apply. As other low-pass and high-pass filters, the median filter does a convolution on the image but with different window operator. Unlike average filter and Gaussian filter, median filter tends to preserve the edge while smoothing images. So it becomes a good candidate in our malware image analysis.

2.5.2.5 Frequency Distribution

Based on the concept of JPEG compression, we can generate a frequency distribution for an image by taking the quantized DCT coefficient into account. Recall that in JPEG compression, we apply the DCT to acquire a coefficient matrix for each 8x8 pixel block. Then we quantize the matrix to reduce less informative data, particularly for the low-frequency part. Finally, we re-order the matrix in a zigzag scan manner to place each particular frequency in a descending order. Hence, if we want to get a frequency distribution to represent the whole image, we then collapse each distribution for each 8x8 pixel block to construct an overall distribution where the high-frequency energy part of the image falls into the top position and low frequent part to the end position. In sum, this frequency distribution tells the variance of visual look in terms of frequency regardless of its position.

2.5.2.6 Statistical Features based on Wavelet Transform

Regardless of the heavy mathematical structure of the wavelet transform [27], we are more interested in extracting some statistical features based on a frequency domain. Very roughly speaking, the wavelet transform can be used to analyze the signal in the frequency domain with a temporal information, unlike Fourier transform, it more focuses on the precise frequency analysis. Furthermore, wavelet transform has been widely accepted to use on many image processing analysis.

In our case, we decompose the image to several sub-bands as shown in Figure 17. We then compute some statistical features from each band. For example, we compute the mean pixel value on the coarse band and the variance on the fine band. To increase the resolution of the feature, we tilt each sub-band image into multiple blocks and then apply the extraction accordingly. One reason we extract different feature on the

different band is due to the information gain by the given band. For example, the coarse band provides fewer details but more overviews. Lastly, we concatenate each value to form a feature vector and apply a uniform quantizer Q to produce a length-l vector $x = Q(\text{statistical features}) \in \{0, 1, ..., 127\}^l$

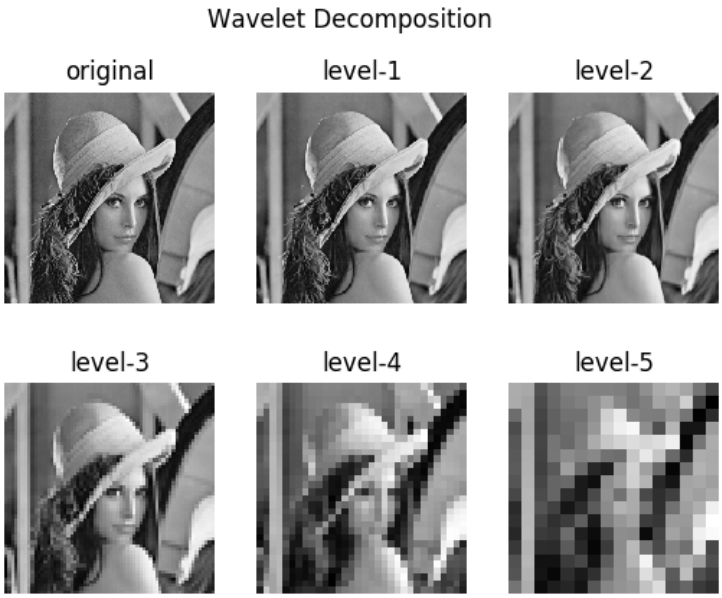

Figure 17: Wavelet decomposed image for each level (sub-bands).

2.6 Support Vector Machines

Among those well-known machine learning techniques, Support Vector Machines (SVM) [28] is a supervised learning and a powerful tool which has been widely used in many machine learning problems due to its efficiency and simplicity. Especially for the classification, unlike an HMM or PCA which typically generates scores, SVM produces the direct classification result. Therefore, SVM is always a good starting tool to address classification problems.

2.6.1 SVM Overview

From the perspective of classification, a good classifier should separate each class clearly. In other words, there are one or more hyperplanes that can split the training samples into the corresponding classes. And SVM comes into the picture. Figure 18 gives an overview. In general, there are four ideas behind SVM as following.

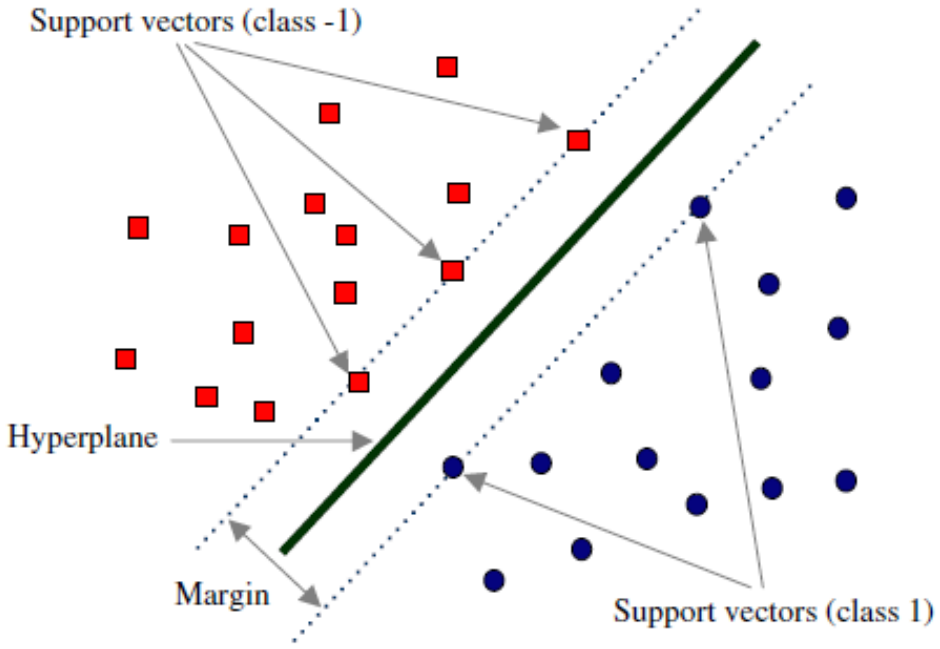

Figure 18: Separating hyperplane with maximizing margin.

- Separating hyperplane

  For typical SVM, we are used to indicating a binary classification. Therefore, one of the goals of SVM is to find a hyperplane that can separate two labeled data. The hyperplane here means a space with 1 less dimensionality than the original space.

- Maximize the margin

  Not only finding a separating hyperplane, SVM tends to maximize the margin between training data and the hyperplane. One of the significant advantages is that it can have more space to classify the data. On the other hand, a disadvantage is the margin could be affected by an outlier, fortunately, SVM has a mechanism to deal with this, that is, support vectors.

- Classify in a higher dimensional space

  Sometimes the data is bare to be separated in its own space or dimension, however, in the context of SVM, it provides a way to transform the original space to higher space in order to get a more clear view for the data.

- Kernel trick

  To transform the data to higher space brings a significant drawback, which is heavy computation. However, as the name suggests, this kernel trick plays an important role to transform the data with much less computation overhead. Figure 19 shows data separation in different view of dimensions.

2.6.2 Training Phase

Since SVM is a supervised learning having labels with 1 and -1 as an example of binary classification. As mentioned before, the goal of SVM is to find a separating

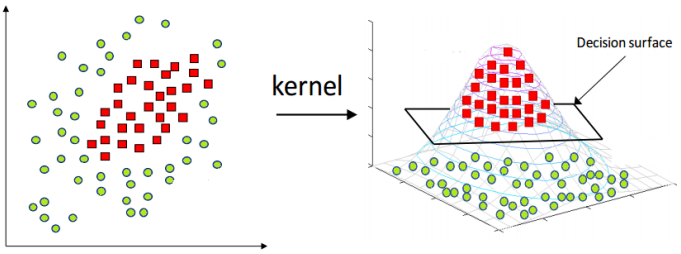

Figure 19: SVM kernel trick.

hyperplane with maximizing its margin to training sample points. As a result, an optimization problem comes into the picture. To be more specific, the problem of SVM training becomes an optimization problem. The general training steps are given below.

1. Given a set of labeled training samples X_i where $i = 1, 2, ..., n$ with the corresponding class in labeled value $z_i \in \{+1, -1\}$.

2. Choose a kernel function K and a regularization parameter $C > 0$, where C indicates the tolerance of the margin.

3. Obtain λ_i and b by solving the optimization problem

$$\text{Maximize} : L(\lambda) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j z_i z_j K(X_i, X_j)$$

$$\text{Subject to} : \sum_{i=1}^{n} \lambda_i z_i = 0 \text{ and } C \geq \lambda_i \geq 0 \text{ for } i = 1, 2, ..., n$$

2.6.3 Scoring Phase

To score the testing sample, by leveraging the hyperplane, SVM simply plug the testing sample in the trained hyperplane and check the direction with respect to the labeling training data.

1. Given a testing sample y and obtain a score.

$$f(y) = \sum_{i=1}^{s} \lambda_i z_i K(X_i, y) + b$$

where λ_i and b is computed from the training phase and s is the number of support vectors

2. Classify the score according to

$$c(y) = \begin{cases} 1 & \text{if } f(y) > 0, \\ -1 & \text{otherwise} \end{cases}$$

CHAPTER 3

Implementation and Result

3.1 Dataset

The dataset we use from [6] is called MALIMG dataset. As Table 3 shows, this dataset consists of total 9,342 grayscale images of transformed binaries from 25 families including family type Wrom, Trojan, Backdoor, PWS, and etc. Figure 21 gives an overview of malware images belonging to various families. The detail of how does an executable file transform from the raw binary to a two-dimensional image can be found in Section 2.2. To construct our benign samples, we use 120 executable files that can be downloaded for free online and passed the antivirus software we have, for example, the 360 Total Security. The benign executable files include PHPSetup.exe, Audio Record Wizard.exe, WindowsDeviceRecoveryToolInstaller01.exe, and etc. To make the samples comparable, we also convert them to grayscale images. By doing this, we obtain 120 grayscale images of benign samples. Figure 20 shows some benign images that are transformed from various executable files.

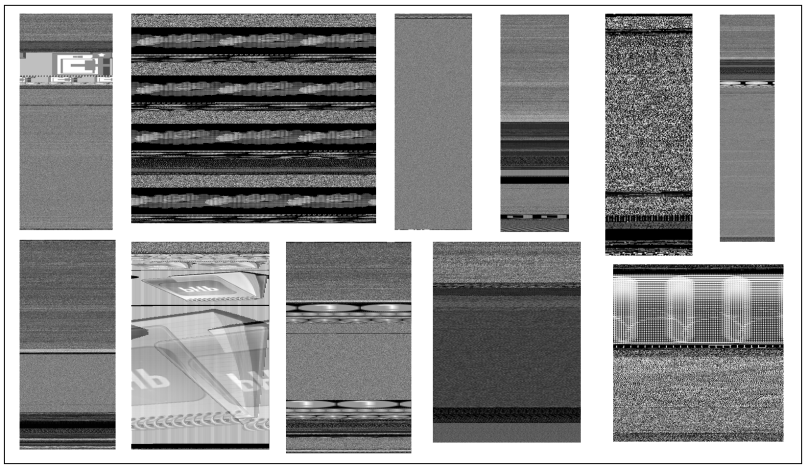

Figure 20: Examples of benign images.

Table 3: MALIMG Dataset.

No.	**Family Name**	**Family**	**No. of Variants**
1	Allaple.L	Worm	1591
2	Allaple.A	Worm	2949
3	Yuner.A	Worm	800
4	Lolyda.AA 1	PWS	213
5	Lolyda.AA 2	PWS	184
6	Lolyda.AA 3	PWS	123
7	C2Lop.P	Trojan	146
8	C2Lop.gen!G	Trojan	200
9	Instantaccess	Dialer	431
10	Swizzor.gen!I	Trojan Downloader	132
11	Swizzor.gen!E	Trojan Downloader	128
12	VB.AT	Worm	408
13	Fakerean	Rogue	381
14	Alueron.gen!J	Trojan	198
15	Malex.gen!J	Trojan	136
16	Lolyda.AT	PWS	159
17	Adialer.C	Dialer	125
18	Wintrim.BX	Trojan Downloader	97
19	Dialplatform.B	Dialer	177
20	Dontovo.A	Trojan Downloader	162
21	Obfuscator.AD	Trojan Downloader	142
22	Agent.FYI	Backdoor	116
23	Autorun.K	Worm:AutoIT	106
24	Rbot!gen	Backdoor	158
25	Skintrim.N	Trojan	80

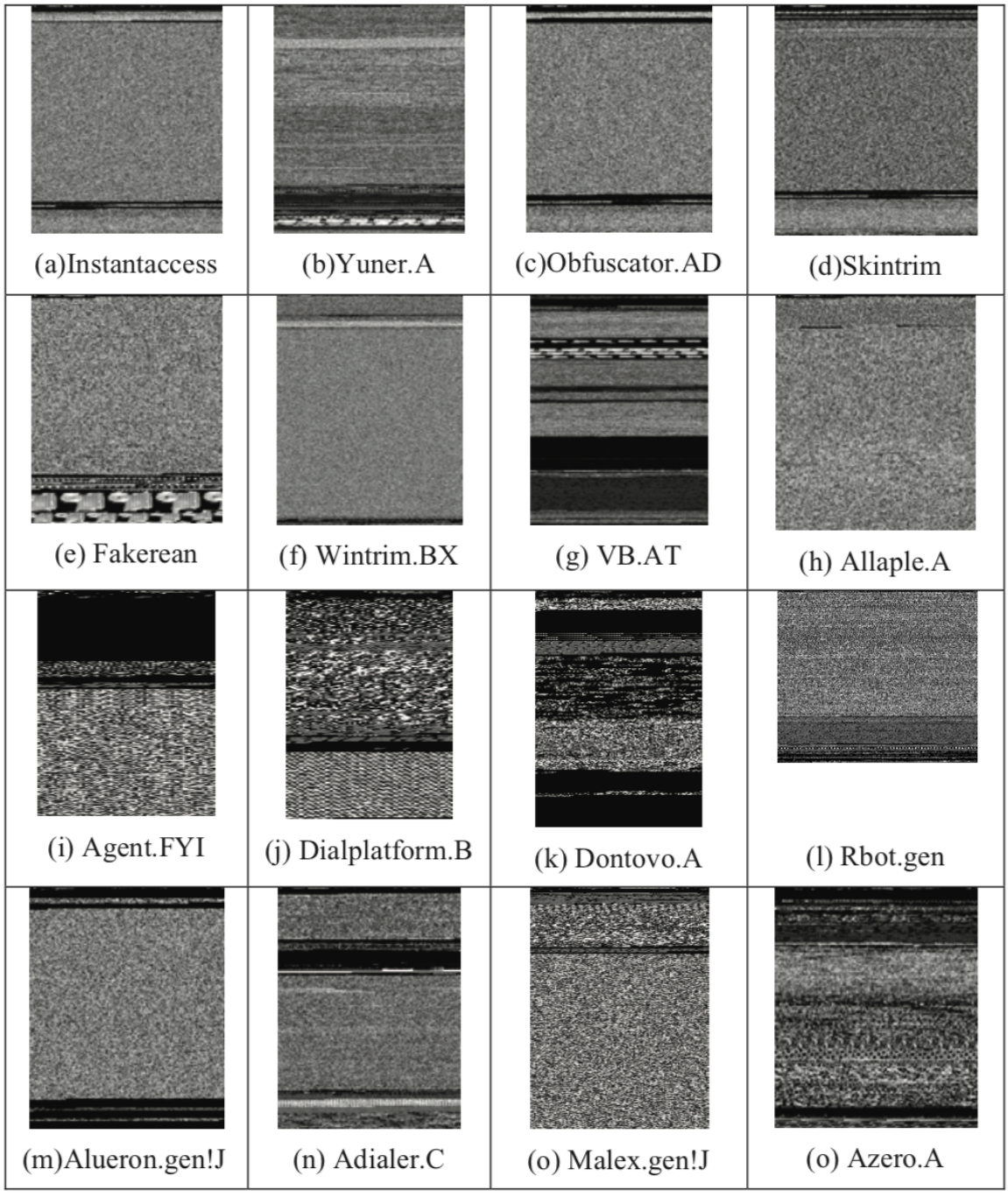

Figure 21: Malware images belonging to different families.

3.2 Classification

Before we dive into the detection process, it is always worth to evaluate how well the selected features are and how good the model is. General speaking, if we can properly classify the malware families with the selected features and classifiers, those features and models are then played as a good representation in some sense.

3.2.1 Machine Learning-Based

From a high volume of machine learning techniques, we pick SVM as our classifier due to its robustness, simplicity, and efficiency. Therefore, in this section, the discussion will be covered from feature selection to the classification result.

3.2.1.1 Feature Selection

In the machine learning-based approach, we use 2 global and 4 local features in the experiment. Those are LBP, HOG, Horizontal-Edge, Pixel Intensity, Contrast, and Median Value. Although there are many types of feature in image processing, the reason we only pick total 6 kinds of features in our experiment is that malware image provides limited information. All we can perceive is not really human readable, for example, most of the images consist of noise and line segments. As a result, we do not use a feature such as color, saturation or SIFT, which is commonly used in object detection.

3.2.1.2 Parameter Tuning

In our parameter tuning, we classify all 25 families with smaller data set by five-fold cross-validation for each type of features and take as much parameter sets as possible. The goal is to find the "best" parameter set in terms of every individual

type of features. Figure 22 shows the accuracy with respect to different settings in the certain type of feature for the global features, and Figure 23 shows the same result for the local features.

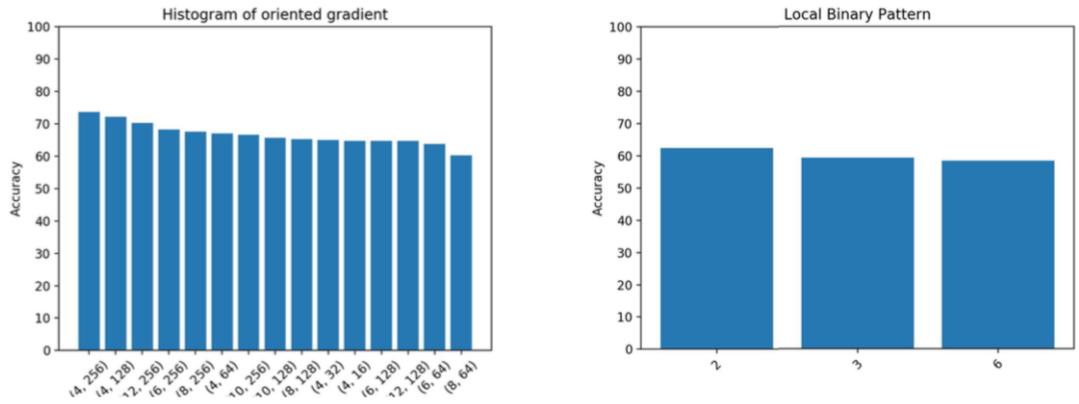

Figure 22: Classification accuracy using the local feature.

It is worth noting that in Figure 22, the global features do not perform well in the case of malicious images, the highest accuracy is less than 80%. Whereas the local features in Figure 23 outperform the global one, they nearly achieve 90% accuracy. And this can be explained by the fact that the malware images are dominated by noises and lead to a bad accuracy.

3.2.1.3 Feature Reduction

In the context of classification, we would like to reduce those unrelated feature dimensions to keep a good consistency. As a result, we proceed a feature reduction process in two steps, Univariate Feature Selection (UFS) and Recursive Feature Elimination (RFE).

1. Univariate Feature Selection (UFS)

   In this step, we take each type of features to feed into our classifier then keep the

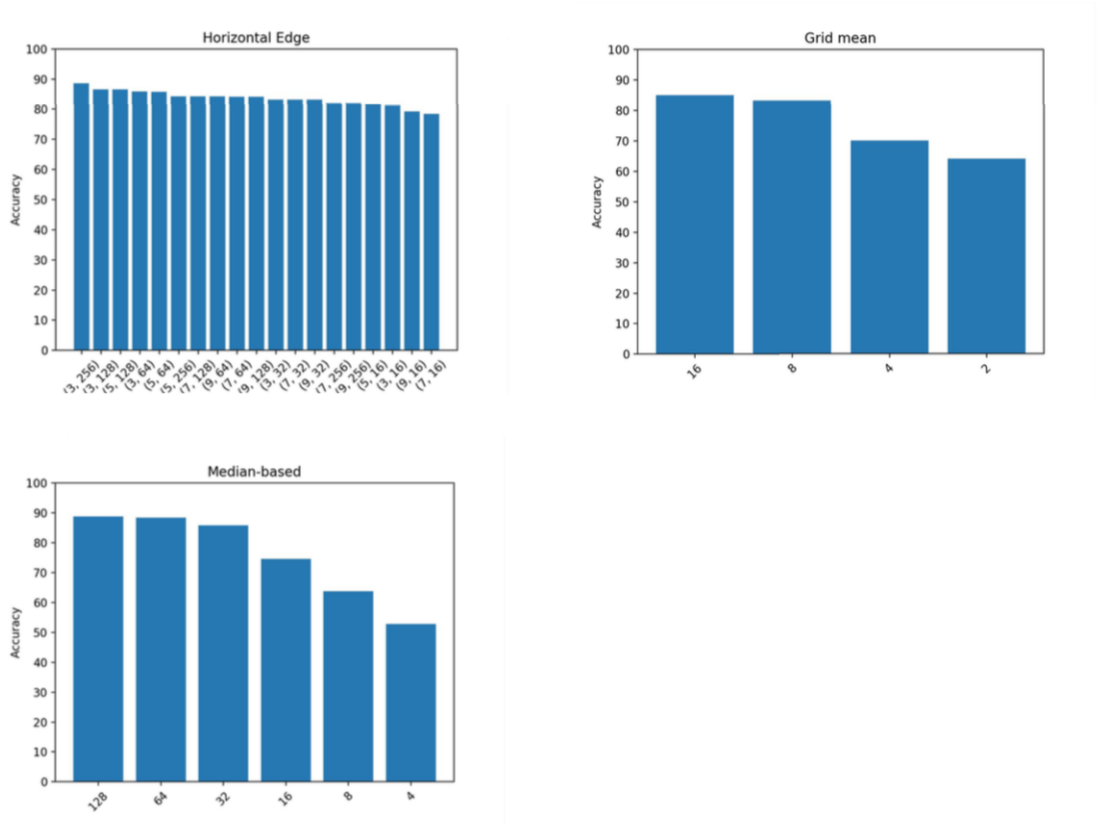

Figure 23: Classification accuracy using the global feature.

best one in terms of accuracy, then based on the selected one, we combine with the rest features to generate a two-types feature up to total 6. Figure 24 depicts a bar chart for the experiment. Surprisingly, one of the features outperforms the others, and it is the horizontal-edge feature.

2. Recursive Feature Elimination (RFE)

   According to the result from UFS, we pick the feature with a highest accuracy as an input in the RFE process. But the length of dimensionality is still too high, for example, 256 dimensions in our case. Thus, we run a feature reduction process. RFE is an iterative process to eliminate feature dimension. In brief, it removes a dimension, re-trains the rest of dimensions, and removes so on and so

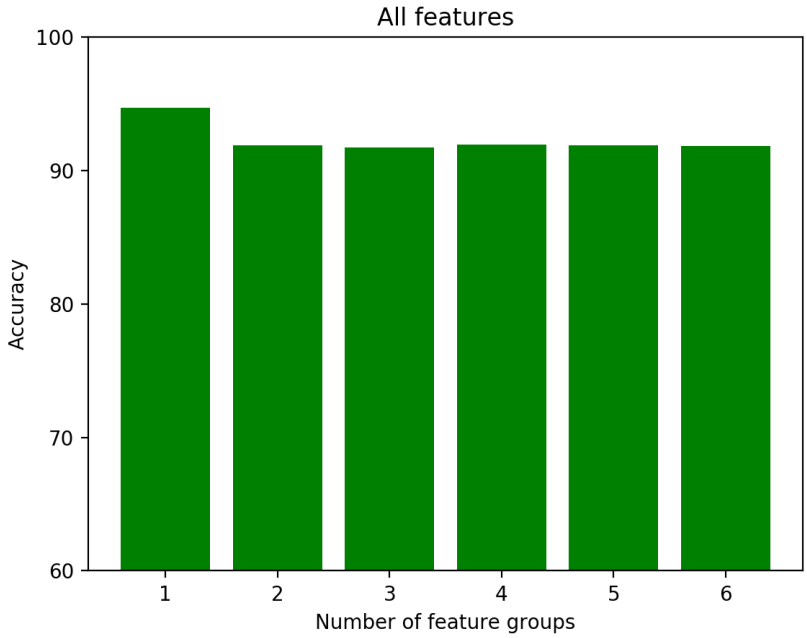

Figure 24: UFS result of all 6 types of features based on malware accuracy.

forth. The result can be seen in Figure 25. The accuracy is getting lower when we reduce the dimension to 20% of original dimensions (50 out of 256), and this is because the horizontal-edge is designed to represent the whole structure of an entire image. Thus, even though cutting the dimensions to 50 can reduce more space, we pick the one with the maximum accuracy in the number of 169 dimensions.

3.2.1.4 Result

In our classification experiment, We achieved 92% overall accuracy on 25 families with five-fold cross-validation using reduced feature in SVM classifier. Figure 26 gives a confusion matrix view for more classification detail.

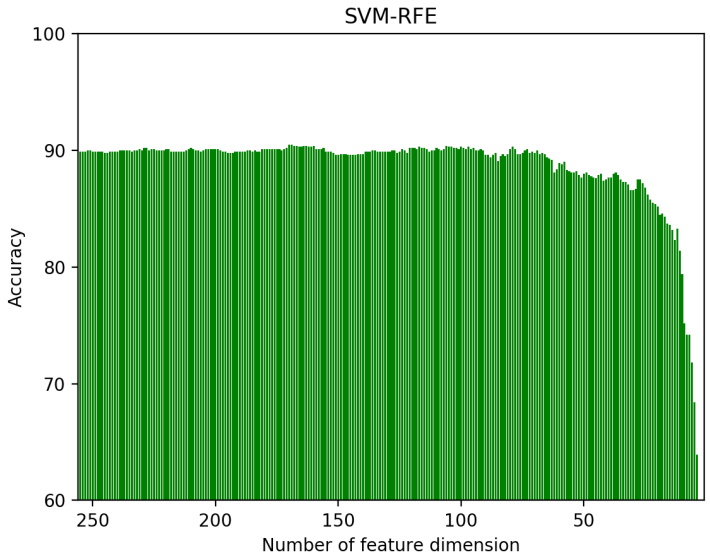

Figure 25: RFE result on horizontal-edge feature.

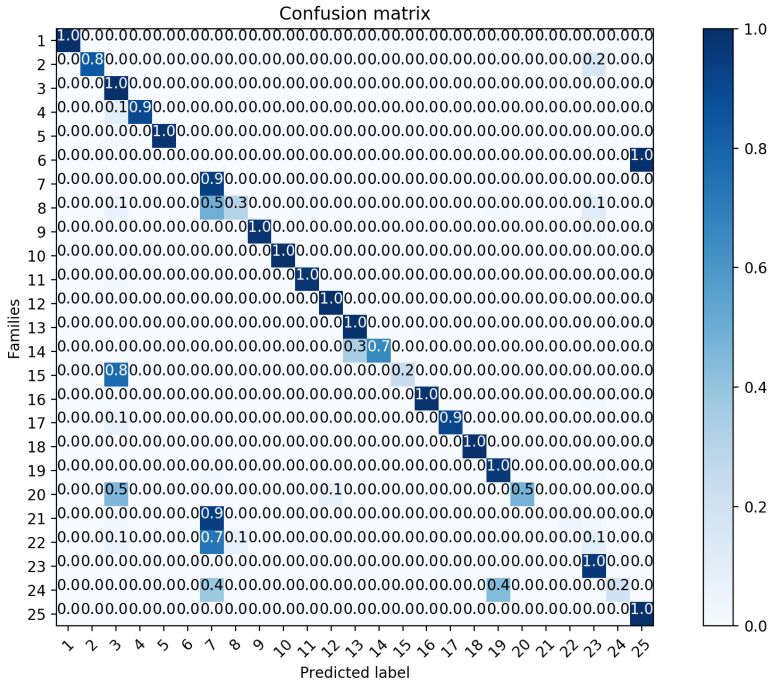

Figure 26: Confusion matrix of classifying all 25 families using reduced horizontal-edge.

3.2.2 Robust Hashing-Based

Recall that in Figure 5, a general robust hashing has two major steps, feature vector extraction and compression. In this project, we implemented two approaches from [4] and [16], and then we conducted an improvement based on those two methods. The following sections will discuss each approach in detail from the step of feature extraction to classification.

3.2.2.1 Error-Correcting Approach

This approach is based on [4], using a wavelet feature and a error-correcting decoder. The general idea is that extract some statistical information (features) from different frequency bands and then put those features altogether by eliminating small difference leveraged an error-correcting decoder.

Under this approach, with 25 families and five-fold cross-validation we achieve 79% average accuracy and 90% while taking 19 families into account. Figure 27 gives more detail. Specifically, we use a 2D Haar-wavelet transform with 5 levels decomposition and apply an uniform tilting for each sub-band to form a feature vector. And then we use a scalar quantization to normalize the feature vector with 128 scales. After all features are acquired, we choose Hamming(7, 4) decoder to correct the sequence of quantized 7-bits value to generate a vector representing an image from one family.

- Implementation

  We compute a representative hash value for a certain family by taking the mean hash values from the training set. And therefore, we will have multiple keys to form a model to indicate which hash value should belong to which family. For

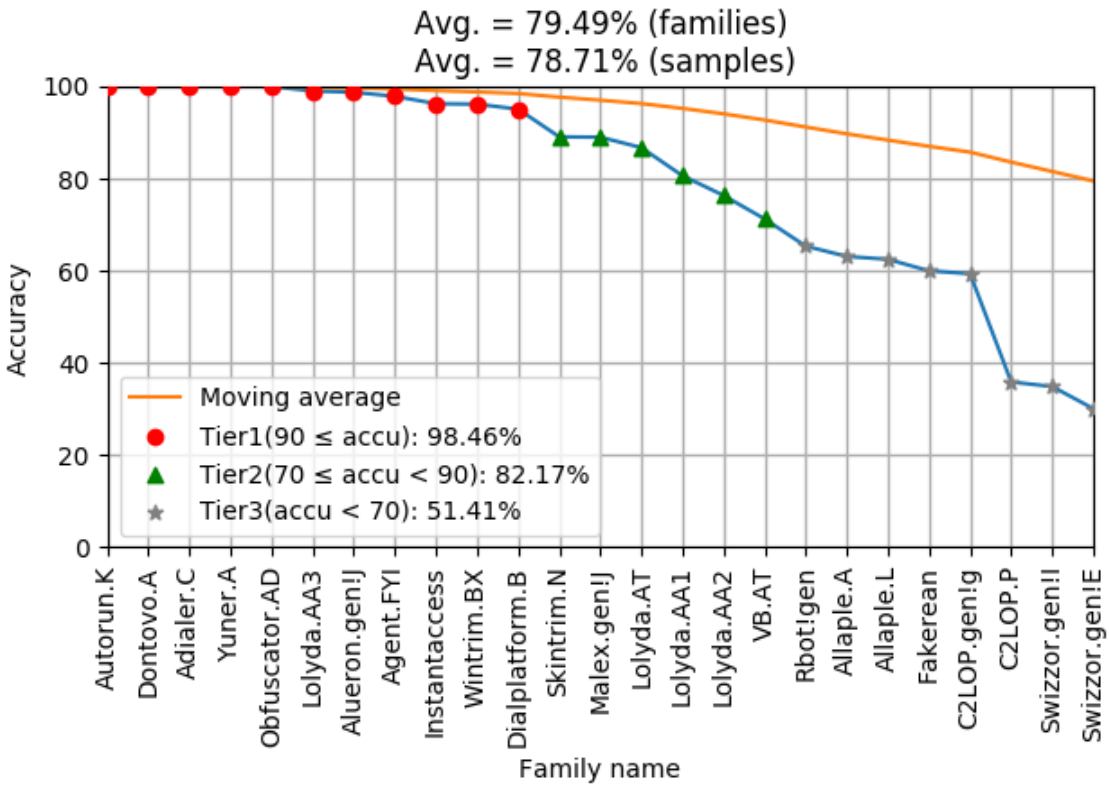

Figure 27: Classification result with error-correcting approach over 25 families.

example, we classify a sample I to I_i if $distance(I, I_i)$ is the minimum Euclidean distance for $i = 1$ to 25.

Furthermore, Figure 28 gives another view while using wavelet features in our application. For the first and the second (from left to right) bin, it shows that the fewer wavelet level is used, the fewer information is gained, therefore the accuracy is compromised. However, if we fix the wavelet level while increasing the number of tilting blocks, the accuracy is increasing as shown in (5, 5) setting to (5, 10) in the same figure. In other words, the result suggests that the more wavelet levels and blocks not only increase the chance of acquiring the information but also improve the accuracy.

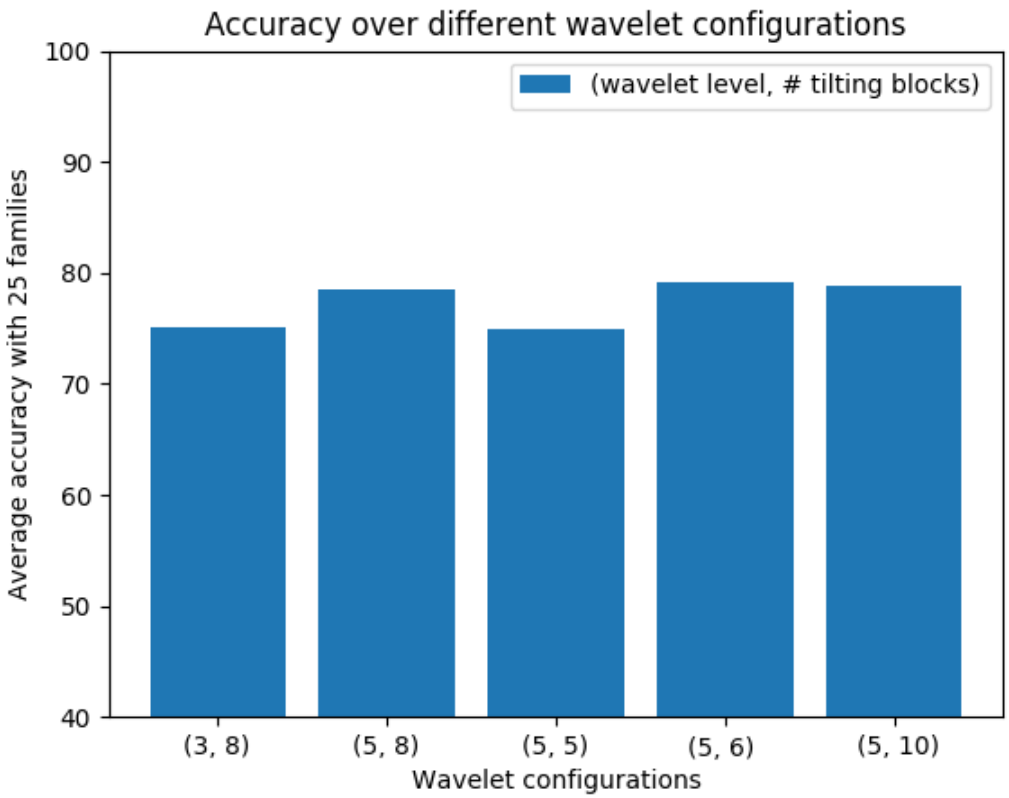

Figure 28: Accuracy in different wavelet transform level.

3.2.2.2 Distributed Coding Approach

From the idea of distributed coding [16] and its diagram Figure 29, we know that we could put a side information (which is dither in our case) as another secret key in a hashing and produce less side effect to the hash value. And from [29], we implement Wyner-Ziv encoder by using a JPEG compression along with Hamming code to produce the syndrome as the final hash value.

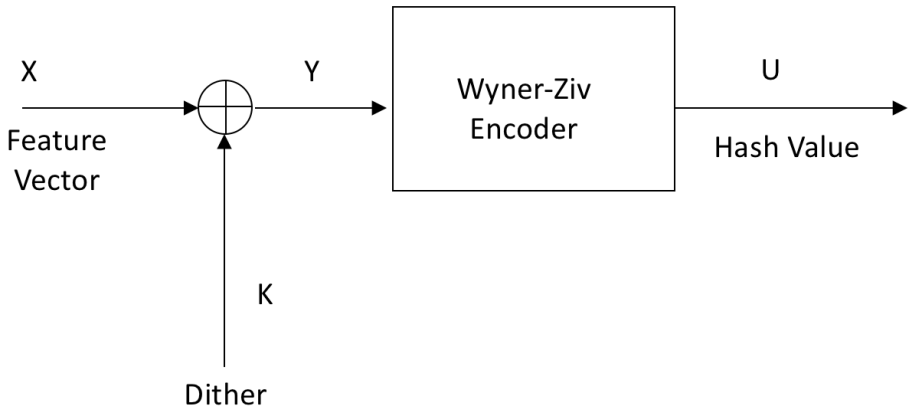

Figure 29: Overview of distributed coding scheme.

Overall we achieve 83.34% classification accuracy over 25 families by using five-fold cross-validation. Compared with the error-correcting approach, it improves 4% accuracy. However, the dithering process is not a cheap work, it dramatically reduces the efficiency more than 10 times. And therefore, we isolate the dithering step to keep good efficiency with less impact on the accuracy, that is, we are able to maintain almost 83% accuracy and a consistent performance from the previous approach. As Figure 30 shows, without dithering, the overall accuracy is 82.63%, and 90% accuracy can be reached if we only discard 3 families over 25 families.

- Implementation

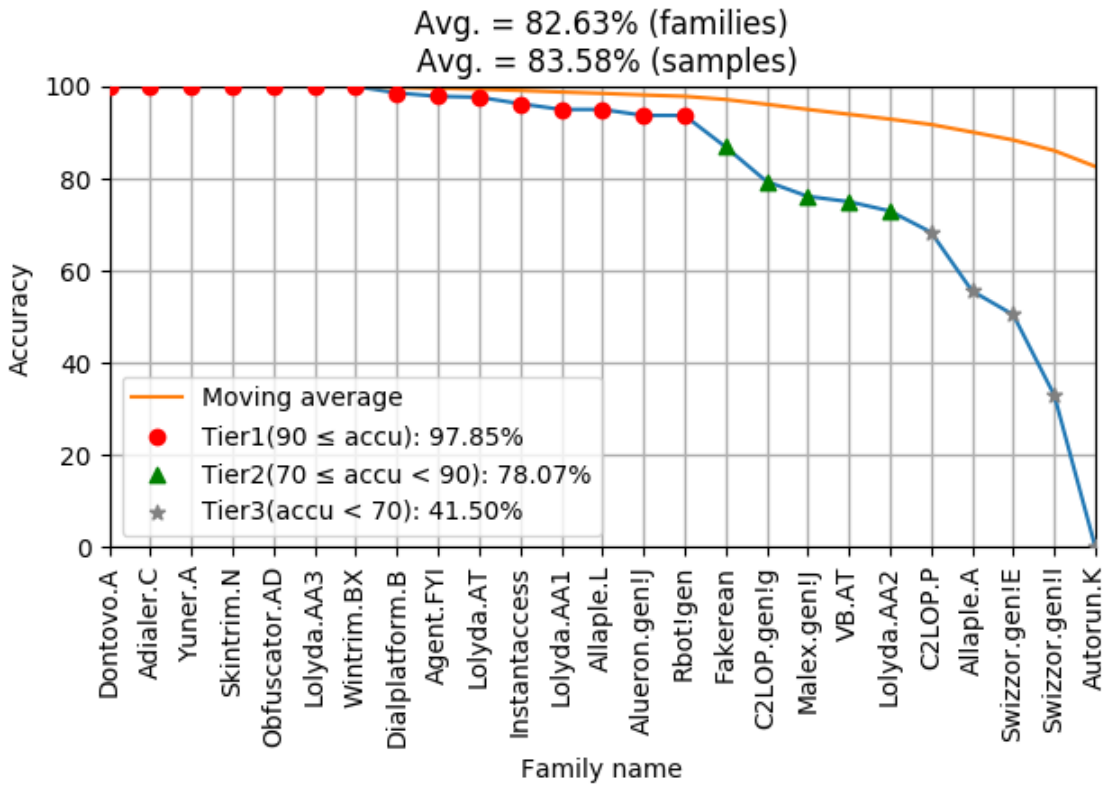

Figure 30: Classification result with distributed coding approach over 25 families.

There is one adjustment we put in this approach. In feature extraction step, not only we take each DCT block to generate a frequency distribution then produce the feature vector, but also we separate a source image into several horizontal segments before extracting features. The reason we do this way is we observe that the major visual difference between each family is the position of those horizontal lines in an image. Hence, we believe that we can extract better information by segmenting some horizontal blocks. Figure 31 shows the result of using a different number of segments and it also supports our horizontal segments assumption. As it suggests, when we uniformly split the image into 8 horizontal segments, the better accuracy can be achieved. The reason is that

more segments will consider more unrelated pixels (noise), and fewer segments will lose the detail from the perspective of global view.

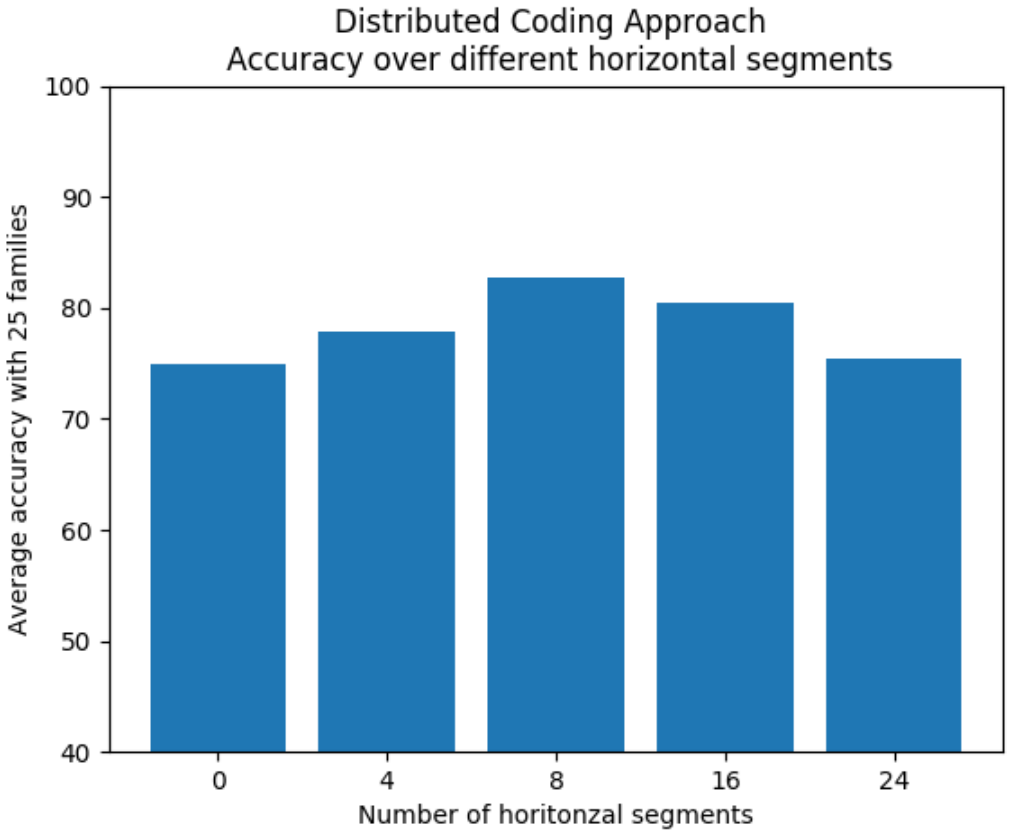

Figure 31: Accuracy with different horizontal segments in distributed coding approach.

After we acquire an extracted feature vector V_i for each family sample, we then use Hamming(7, 4) to encode and produce a syndrome by concatenating the 3 syndrome bits to form a length l string S_i, where $i = 1$ to the number of training samples in a family. For all S_i we then pick and append the majority symbol (0 or 1 in our case) at the aligned position within the length l to generate our final hash key K_i. Finally, we use Hamming distance function f_h as an evaluation metric to determine the classification. That is, we classify a sample I to I_i if $f_h(I, I_i) < f_h(I, I_j)$ where $i \neq j$.

3.2.2.3 Multi-phase Approach

From the classification result of the previous approaches, we observe that some families are classified well in one approach whereas classified bad in another, for example, family Autorun.k is classified well in error-correcting approach but in distributed coding approach. Given this observation, we decide to combine this two approaches to conduct a new multi-phase approach which is shown as follows.

1. Scoring from error-correcting approach

   Recall that we have a set of hash keys $K_a = \{H_a K_i\}$ which are generated from the error-correcting approach where i is the index of family and a is the index of approach, and we then compute a Euclidean distance for each incoming sample I with K_a to get a set of distance where $D_a = \{D_{ai}\}$. Then, we get a set of similarity scores $S_a = 1/(1 + D_a)$. The higher score it is, the higher probability I will be classified to.

2. Scoring from distributed coding approach

   Similar to the phase1, we replace the part of classification approach with distributed coding. Clearly, we will also get a set of scores $S_b = 1 - D_b$ where D_b is a set of normalized Hamming distance for input I to the stored hash keys K_b.

3. Merging

   This phase will combine the score set S_a and S_b, and then determine which classification result should be used by taking the variance of scores in each set into account. In other words, by selecting the ability of classification for different approaches, multiple choice manner should perform better than just using a single approach. Intuitively, it is easy to pick the best score of them

then determine which approach should be used, however, those scores are not from the same distance basis, that is, the score only correlates to itself within the same approach. As a result, we devise a way to combine those scores by connecting different score bases altogether. To be more specific, for S_a and S_b, we compute a variance which provides information of how "separate" each set of hash keys is. For example, if top 3 score of S_a is 1.0, 0.9, 0.9, and top 3 score of S_b is 1.0, 0.7, 0.4, and then S_b will be chosen due to its high variance.

Figure 32 shows the result, 87.3% average accuracy is achieved over 25 families and we are able to maintain a high accuracy 90% by just discarding one family.

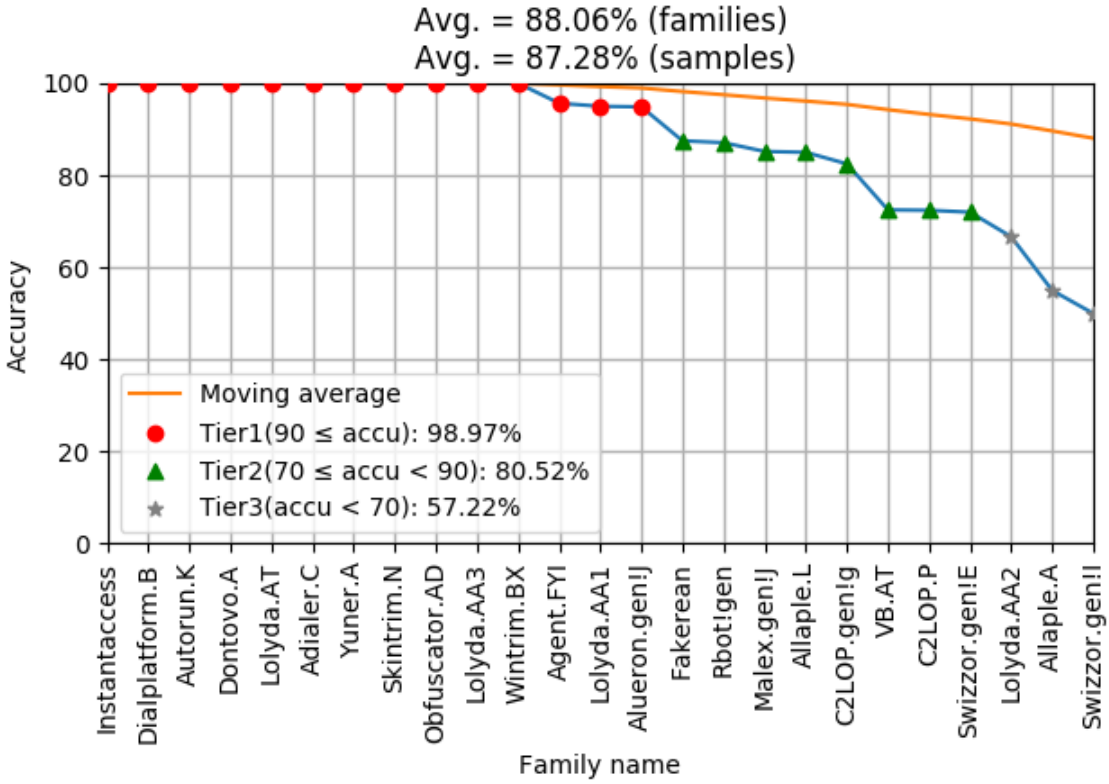

Figure 32: Classification result with multi-phase approach over 25 families.

3.2.2.4 Comparison

In the comparison section, Figure 33 shows the classification accuracy over three different hashing approaches. Obviously, the multi-phase approach is able to select a better approach from the error-correcting and the distributed coding, as a result, it generates a better classification accuracy. It is also worth noting that the distributed coding has 0% accuracy at the family Autorun.K but the error-correcting approach can achieve 100%. After we carefully investigate that family and the scores, we find out there is another family Yuner.A which is very close to Autorun.K. As Figure 34 shows, these two families are perceptually identical and this explains why Autorun.K is classified to Yuner.A in the distributed coding approach.

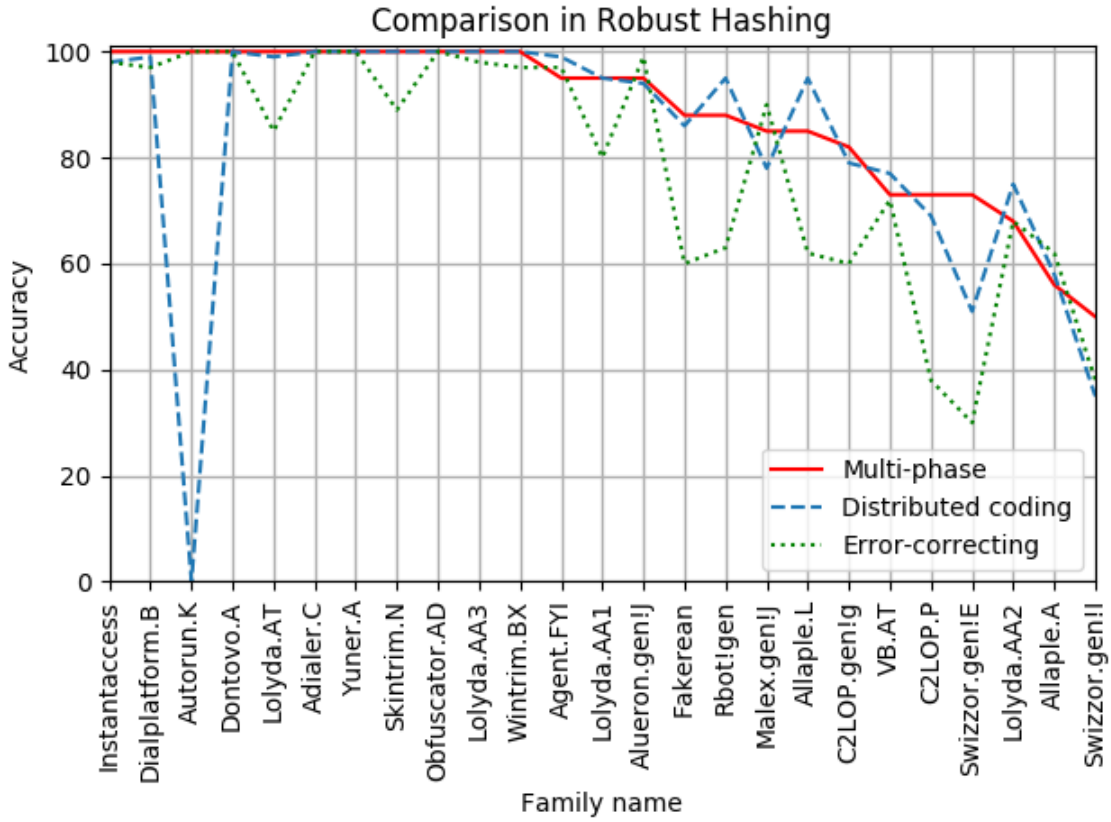

Figure 33: Robust hashing with different approaches.

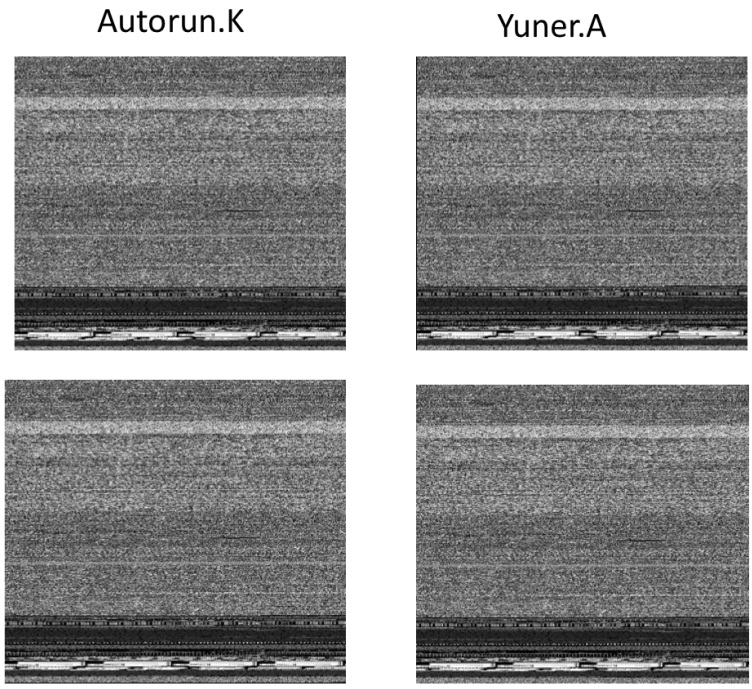

Figure 34: Image view of family Autorun.K and Yuner.A.

We also want to examine what is the outcome if we mix up the hashing approaches and the features plus the h-edge which is well performed in our machine learning-based approach. So far we have two channel coding types, Hamming decoding (error-correcting) and Hamming encoding (syndrome), and two features, frequency distribution and statistic wavelet. And now we want to examine another feature, h-edge, which is the selected feature from our machine learning based approach in Section 3.2.1, to see if it still works in a robust hashing manner. We carefully experiment all of these three features and two channel coding types while picking the best parameter configuration on each experiment round.

Figure 35 gives the overall comparison with regard to the average accuracy over 25 families. As a result, we have the following two conclusions. First, the h-edge does not perform well with the use of robust hashing which is because, in the machine

learning-based, SVM has support vectors whereas in robust hashing-based it takes the whole feature vector into either decoder (error-correcting) or encoder (syndrome). That is, the channel coding process considers a whole picture of view in the extracted features. Second, the error-correcting approach is less sensitive to the features. That could be explained by the fact that error-correcting is designed to eliminate noise (or the difference between features) such that no matter what features are coming, a certain among of noise will be removed. On the other hand, the syndrome does nothing for correcting or eliminating noise, what it does is picking 3 important bits from the original source data. That is, it is expected to reflect robustness of selected features.

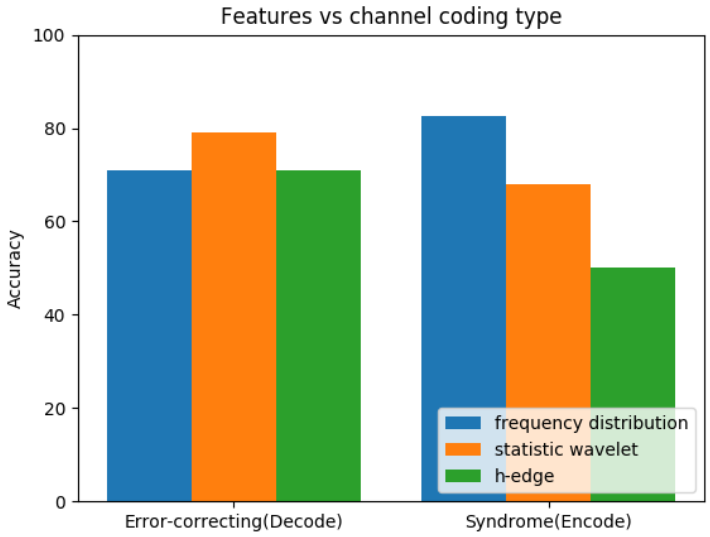

Figure 35: Comparison of different features and channel coding types.

3.3 Detection

In this section, we discuss how we handle the benign case with the existing models and two major approaches from our previous classification architecture. Furthermore, we then show the detection result by the view of ROC curve and AUC table.

3.3.1 Machine Learning-Based

This approach is based on our multi-class classification model in the previous section. To achieve the detection we simply treat all families as one big family, called malware. The idea behind this approach is by using a method proposed by [30] which estimated a probability of each sub-model. That is, each testing sample will output N probabilities where N is the number of sub-models. Given all training samples, we then expect to find a minimum probability which is being correctly classified for each family and that implies the chance of given sample will be classified to which family. For example, in the training phase, if we set those minimum probabilities as the threshold for each sub-model, the sample will be determined as a certain malware family if the classifying probability is larger than the minimum threshold that we have trained. If the classifying probability is smaller than all of them, it will be treated as benign. As a result, we will have a probability as the threshold for all training families. Figure 36 shows the minimum and maximum correctly classifying probabilities within each family.

Also, as Figure 36 suggests, however, some sub-models do not perform well (C2LOP.gen!g, C2LOP.P, Swizzor.gen!E, Swizzor.gen!I). Based on the higher minimum probability indicates the better inner classification, we claim that those low probability sub-models do not contribute that much. In other words, those models could harm the overall detection rate.

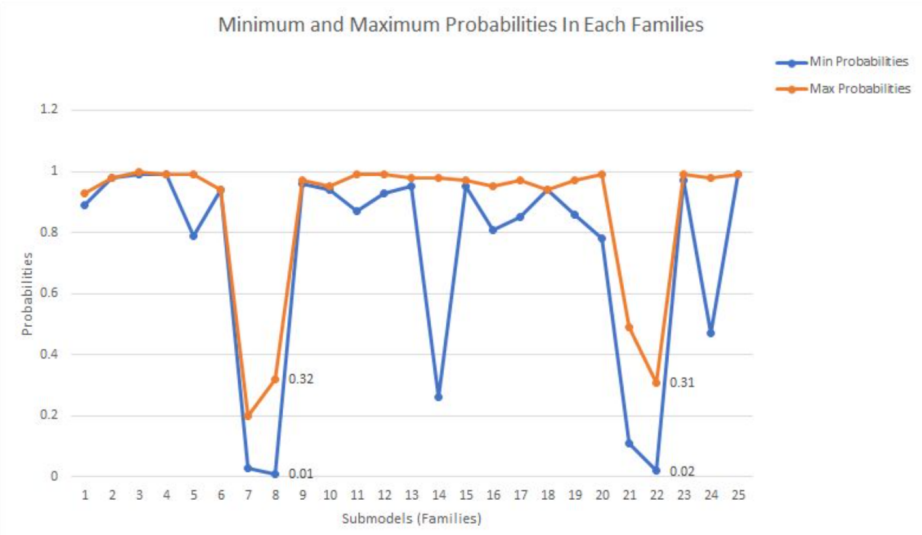

Figure 36: Minimum and maximum probability in each family.

A similar conclusion can also be made from Figure 37, which gives the number of families of which the minimum probability is larger than a certain threshold. The higher the threshold we set, the fewer families satisfy, the better the benign detection accuracy rate will be. As we can see from the figure, the benign detection accuracy rate drop to almost 0 from 0.78 when family numbers increase from 22 to 25.

To better prove our deduction, we experiment the benign accuracy in the sense of ranked models. Similar to UFS, we select the top-k well-performed families as our training set, and Figure 38 shows the result. As we expected, it achieves 91% benign accuracy when 21 families are used. It is worth to note that the detection rate is based on the minimum probability of each trained model being set to be the threshold. The accuracy of malware sustains a very high rate by eliminating the four sub-models that do not perform well.

From Figure 38 we are more interested in the set of 25 families, 24 families, and to the 18 families. In the following experiment, we compare different approaches for determinng the threshold in the detection. First, we treat all sub-models evenly and

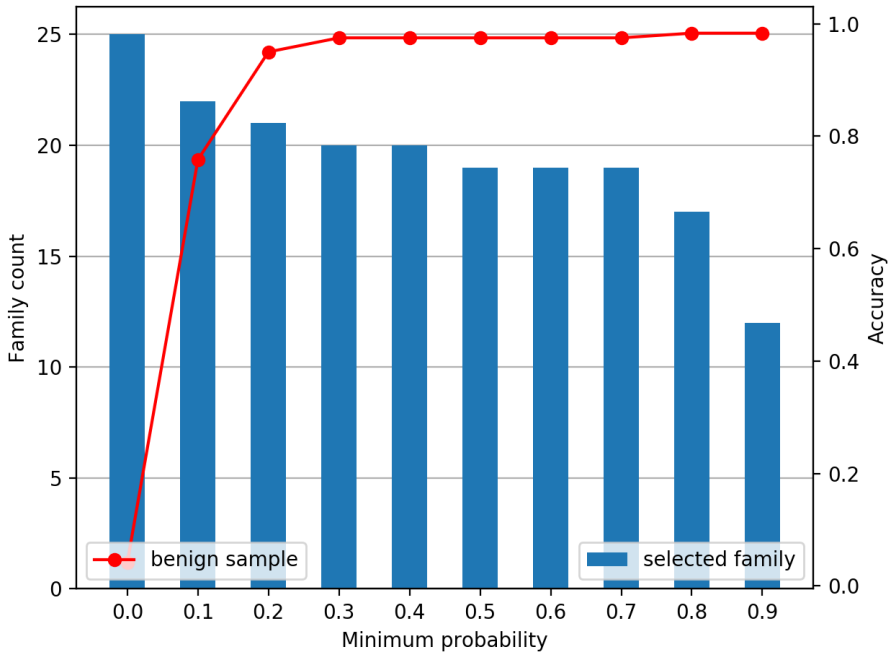

Figure 37: Relationship between the threshold and the number of families.

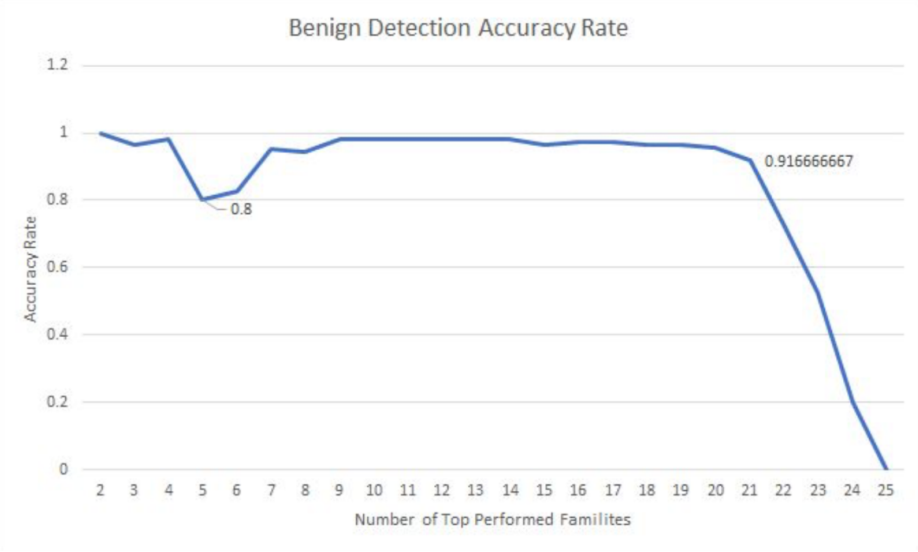

Figure 38: Benign detection accuracy over top k performed families.

give a uniform threshold for each of them. As we concluded before, each sub-model might contribute unevenly. Therefore, in the second approach, we use minimum probability as the base for each family. And in the third approach, we want to

examine the mix case on the first approach and the second approach, that is, we lower each threshold from the second approach by two standard deviations. Figure 39 and Figure 40 reveal the ROC curve and Table 4 shows the AUC results.

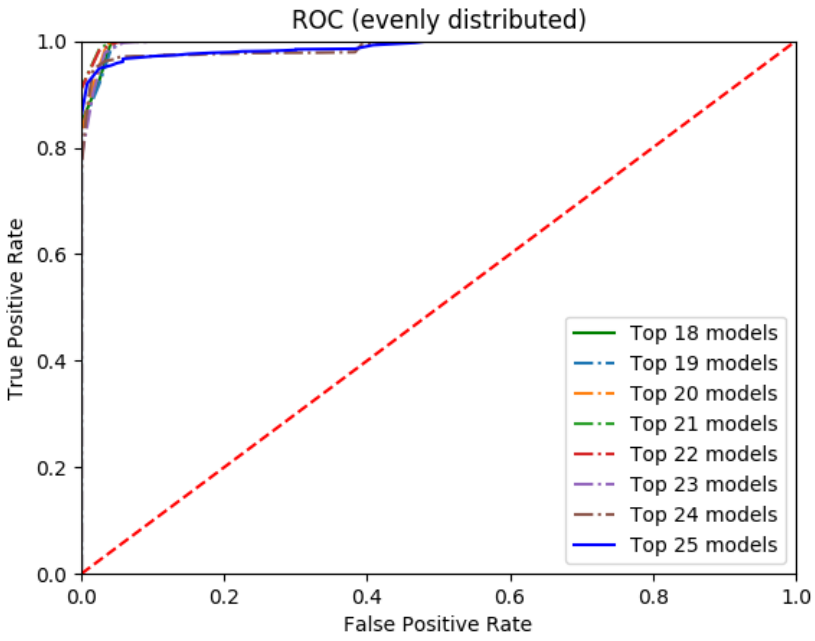

Figure 39: ROC of malware detection by using SVMs with the threshold is evenly distributed.

Table 4: AUC results with the different threshold approaches.

Approach	Number of sub-models							
	18	19	20	21	22	23	24	25
Uniform value	0.996	0.996	0.996	0.998	0.998	0.995	0.987	0.988
Lower minimum probability	0.984	0.986	0.972	0.922	0.794	0.768	0.791	0.801
Minimum probability	0.986	0.989	0.984	0.975	0.904	0.830	0.793	0.756

Clearly, if we use a uniform threshold for all sub-models, the detection rate outperforms the separated one. This can be explained that the generated probabilities of most samples are higher the threshold in an uniform thresholding manner. Also, when

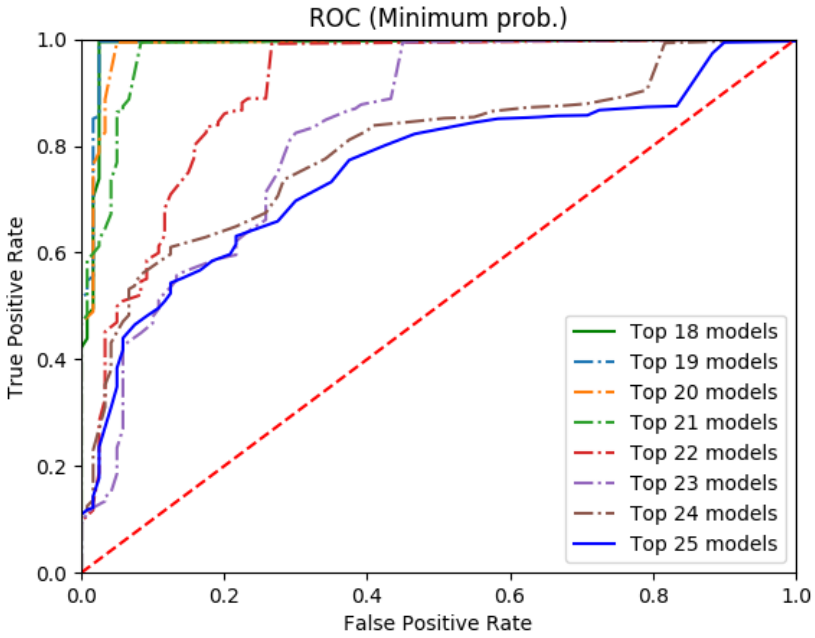

Figure 40: ROC of malware detection by using SVMs with the threshold is using the minimum probability.

we use the minimum probability as the threshold, for the low probability sub-models, there are more zooms to increase the threshold and it is likely to be misclassified as benign, hence the true positive rate drops. In other words, as we discard those sub-models, meaning we tend to keep high probability sub-models, the range of the threshold in each sub-model is more stable.

Given the high threshold (probability) for most of our samples, we want to see what happen if we loosen the threshold to make it lower than the minimum probability but keep their individual impact of each sub-model by subtracting two times of standard deviation on the training probabilities. The result can be seen at Figure 41. As we concluded, discarding some low probability sub-models can improve the accuracy, but the AUC does not change that much in that the samples of mid-high probability sub-models have higher variance such that the AUC is compromised.

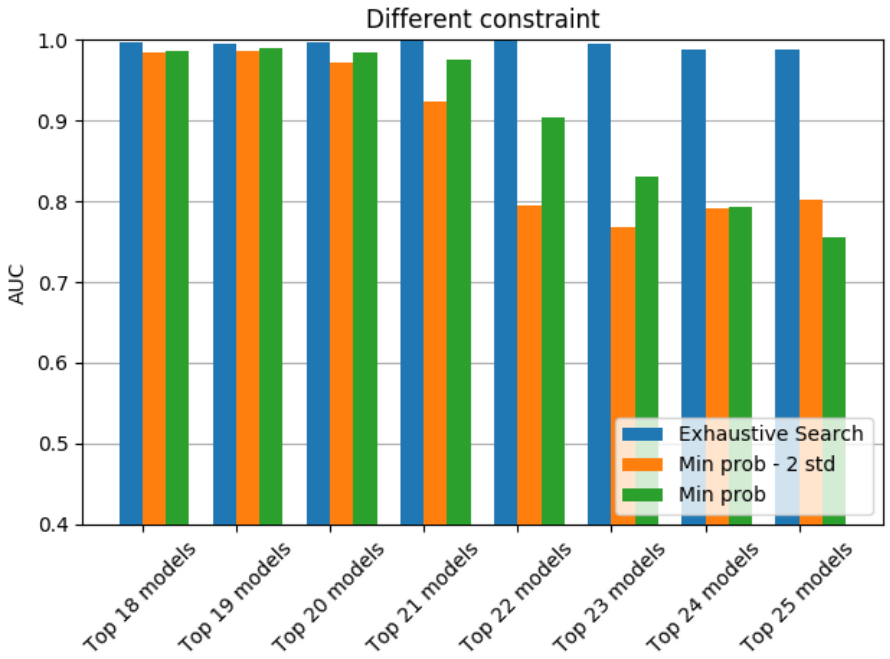

Figure 41: AUC view of different threshold approaches.

Although the approach of using individual weight is worse than using the uniform weight, it is still worth to construct these two models to generate a two-pass detection process. In the first pass, we can use the model with the uniform weight to easily separate benign and malware. If the sample is detected as malware, it will be fed into the second pass, which uses the weighted models. The benefit of using the two-pass model is we can classify the malware more precisely. For example, we can give a strict and loosen threshold for high risk and low-risk family respectively.

3.3.2 Robust Hashing-Based

In this section, we discuss how we use robust hashing to detect malware. Since we already have a hash secret key for each malware family, we can treat all secret keys in all families as a one malware family. If there is a new sample whose hash value is close enough to any family, then say it is malware, otherwise benign.

Instead of comparing the hash value and the secret key directly, we use the score that comes from our error-correcting and distributed coding approach to determine if it is malware or benign. Recall that this score indicates how close to a certain family the sample is. It is worth to note that the reason we do not use the score from the multi-phase approach is that the basis of the score could be vary such that there is no single threshold can determine.

In our experiment, we detect malware based on three robust hashing approaches that we have discussed previously. In order to visualize the detection result, we use a ROC curve and an AUC to represent the robustness of robust hashing on the malware detection. Furthermore, we are also interested in what is the impact of just using top-k well-classified families, such that it can be carefully compared with the result in the Section 3.3.1. Figure 42 shows the best detection rate while using error-correcting approach in terms of ROC curve and AUC. Figure 43 and Figure 44 show the ROC curve of using distributed coding and multi-phase approach respectively, and Table 5 gives more AUC result from three different hashing approaches.

As we concluded in the classification section, multi-phase should have the ability to select the best approach to detect or classify, and our result shows it performs better than the other two approaches. However, there is only one difference in the detection, that is, we use two thresholds for determining which score should be considered.

The reason why error-correcting approach outperforms to distributed coding is that the error-correcting has more abilities to handle unknown samples, and its wavelet decomposition serves as a better feature due to its variety and statistical property in different image sub-bands. In contrast, the distributed coding approach relies more on its extracted features due to the lack of correcting step, even though its classification result is slightly better than the other.

Table 5: AUC results with the different hashing approaches.

Approach	Number of sub-models							
	18	19	20	21	22	23	24	25
Error-correcting	0.814	0.782	0.764	0.754	0.744	0.734	0.731	0.720
Distributed coding	0.732	0.711	0.681	0.677	0.677	0.663	0.649	0.649
Multi-phase	0.810	0.779	0.763	0.745	0.734	0.726	0.722	0.720

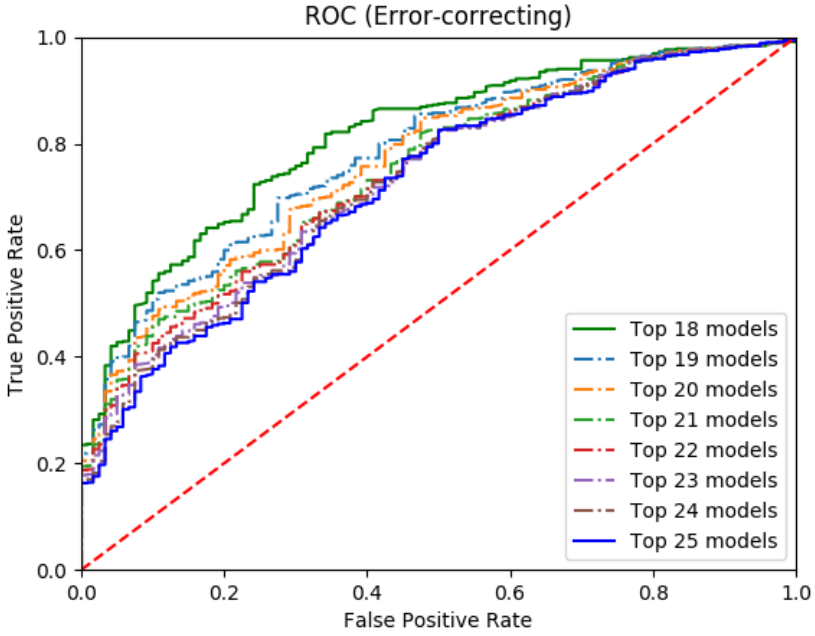

Figure 42: ROC of malware detection by using error-correcting approach of robust hashing.

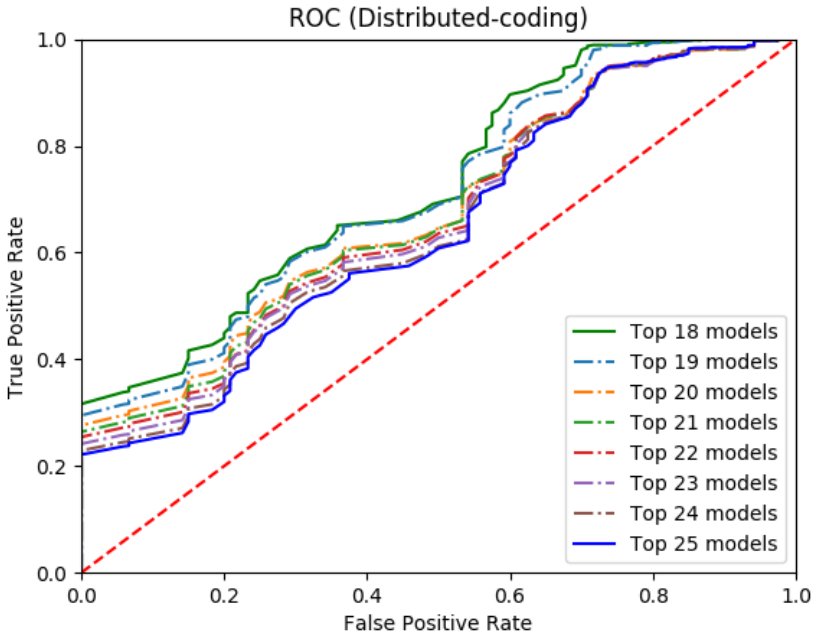

Figure 43: ROC of malware detection by using distributed coding approach of robust hashing.

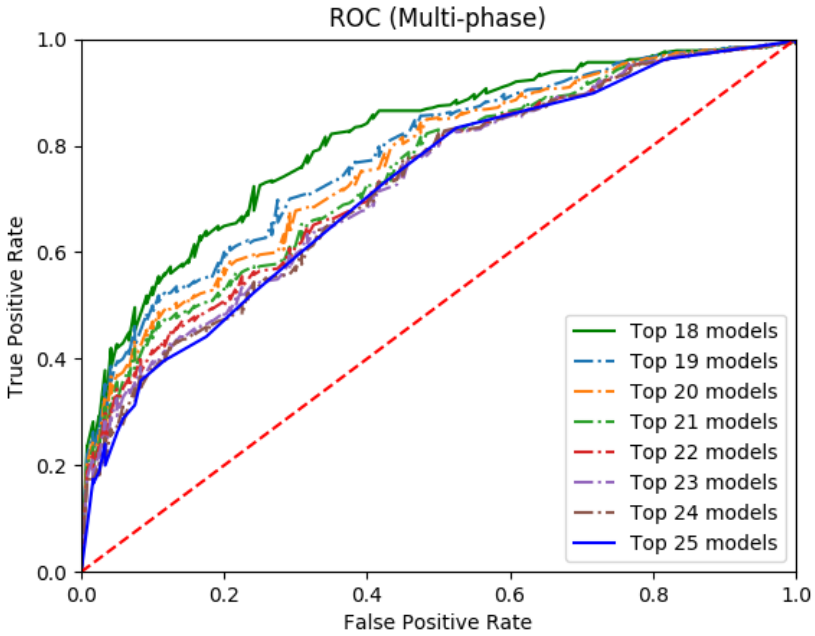

Figure 44: ROC of malware detection by using multi-phase approach of robust hashing.

3.4 Discussion

3.4.1 Machine Learning-Based

The first approach we implement is by using a supervised learning classifier which we pick SVMs, with extracting some image features based on the observation of input images. This also includes feature selection and feature reduction. In our experiment, we carefully select features from the perspective of global and local, and we find out global feature performs better than the local. Furthermore, there is only one feature, h-edge, stands out of total 6 features by running a univariate feature selection, and achieve 92% classification accuracy over 25 different families. We then move on to the detection phase based on our SVM model.

In the detection step, we need to consider another sample set, the benign set. But there are not many similarities in between, for example, the original malware family has many similarities in its line structure. As a result, instead of training the benign set into our malware model, we leverage a generic multi-class SVM scheme to generate a probability for each family of being correctly classified. From this manner, we are able to determine a sample should be classified to a specific family, or classified to benign if none of the probability or threshold is held.

At the first stage, unfortunately, we test all 25 families and the benign set, the detection accuracy for the benign is almost 0%. Meanwhile, we find that there are some families or sub-models do not contribute that much because of low probability they have. So we do another experiment to see the accuracy by discarding the family or sub-model that theoretically contributes less in the whole picture. Finally, we get an AUC 0.756 over 25 families and AUC 0.986 over 18 families. And we relax the assumption that each sub-model has its own contribution in the overall detection scheme, as a result, we get AUC 0.988 over 25 families.

3.4.2 Robust Hashing-Based

Using a robust hashing on malware detection is a novel idea. Through our implementation and experiment, we are able to hash each malware sample into a hash value, or even to generate a secret hash key to represent the family. By comparing with the hash value and the secret hash key, we then can do classification problems under the definition of robust hashing. In our work, we implement three robust hashing methods, error-correcting, distributed coding, and multi-phase. The first two are partially from the previous work. We also use the feature from the previous work as a comparison. Those features are wavelet-decomposition and DCT coefficients. However, due to the application and the specific type of input images that we use, it is not trivial to compare our result with the previous work. Instead, we take the previous work in our experiment as the base to be compared with our machine learning-based approach and multi-phase approach.

In the classification, we get 79% accuracy in error-correcting approach and 83% accuracy in distributed-coding approach over 25 different families. We also carefully analyze different configurations of the features in terms of the average accuracy. Finally, we propose a new method, multi-phase, which is generally based on previous two approaches. It has the ability to pick which approach should be used and as expected, it gets improved and achieves 87% accuracy overall. By showing our experiment result, we also delicately analyze the difference between the previous work and our work.

Based on three robust hashing methods that we have implemented, we take the benign set into consideration and deem every malware family as one giant malware family. That is, we hash a benign sample into a hash value and then compare it with 25 pre-hashed secret keys. If it is close enough to one of malware family, we then

label it to malware, otherwise to benign. Under this manner, we get the AUC 0.72 over 25 families and AUC 0.81 over 18 families by using the proposed multi-phase approach.

3.4.3 Comparison

Overall, regarding classification, machine learning-based approach achieves better classification accuracy, 92% over 87% in robust hashing-based. This is not really surprising because, from the perspective of SVMs, its goal is to maximize the margin for different classes of samples. Moreover, in the step of feature reduction, SVMs is able to have a clear view of the features to train a model. In other words, it can inherently eliminate noise or outliers before going through the training process. If the feature is well selected and representative, then the classification result should be really sweet.

However, for the robust-hashing, even though the feature we have extracted is also representative, unlike SVMs, robust hashing does not have optimization step to find the best separation. Instead, it only takes one class at a time into consideration and trying to eliminate the difference between the samples. Another evidence to prove our conclusion is that SVMs has a kernel trick, which lets the classifier to process in higher dimensions in which the separation might be better.

When considering the detection problem, generally it is more difficult than classification problem. Theoretically, we could say the more malware families we have, the harder we can recognize benign sample. Based on this assumption, we gradually discard one family at a time, and the detection accuracy does improve. Again, machine learning-based approach outperforms than robust hashing-base, however, if we carefully check on the ROC curve of the case of minimum probability in SVM ap-

proach and the cases in robust hashing approach, the latter is relatively stable when we discard certain families. This can be explained that when training the SVMs, some well-defined sub-models might dominate the less-representative sub-models. Or it can say those "good" sub-models are compromised by a small portion of the "bad" sub-models. In that case, when we gradually discard families, the detection accuracy dramatically gets improved. On the other hand, the training process in robust hashing is independent of other families. So even when we reduce the number of families, the detection accuracy does not improve real quick.

In sum, those two ways of malware classification and detection are performing not much significant different in terms of the accuracy. But they could be distinct from practical applications. For example, if there are more families come into the picture, in SVMs, it must re-train the model which is not very efficient, and the most important is that the new model could be significantly different from the old model, therefore, the overall model could be unstable. However, robust hashing can simply train a new family for generating its secret key without introducing significant compromise in the previous keys, hence the burden is much less than using SVMs.

CHAPTER 4

Conclusion and Future Work

4.1 Conclusion

In our work, we presented a novel way for malware classification and detection based on robust hashing and carefully compared it with using machine learning techniques.

First of all, based on the malware images we were able to build a classification model by using SVMs. By observing the given malware images, we concluded that the global features outperform the local features and one of the global features, namely, horizontal-edge feature, plays a critical role in our machine learning-based approach, although there were other 5 features that might be good to represent a malware image. We also carefully examined the configurations of each type of features and leveraged UFS and RFE to select the final feature in our classification model.

Followed by the classification model, we conducted a detection method utilized the probability of each malware family that is being correctly classified. Given the probability of each family, we concluded that some malware families are not well-contributed in the detection process, thus, we trained two models with different probability thresholding metrics. The result showed these two models can be used as a two-pass detection process, that is, the first pass will use an uniform thresholding to determine if it is malware, then in the second pass, the model will be used to do a precise classification.

Secondly, we developed a robust image hashing on the malware images with two different kinds of approaches. These approaches also involved other researching

areas such as communication system, information theory, and data compression. We delved into each categories extracting some concept that can help with the robust hashing and finally completed implementing the classification and detection process. We also improved the classification in terms of the average accuracy by devising a new approach in which we consider a score from the two approaches. At the end, we concluded that while the robust hashing-based approach does not exceed the machine learning-based in terms of the accuracy, it is still worth to use the robust hashing-based approach given that a large number of malware families could be a practical situation.

4.2 Future Work

We presented two ways of addressing malware detection, machine learning-based, and robust hashing-based, where those are based on image processing techniques. But there are some elements might improve our detection.

The first one is the features. Given our observation of malware images, global features play an important role in our case, however, there are few families that do not reflect this property. For example, images in family Swizzor.gen!I are not visually identical in terms of the structure. One way to improve this is obviously by combing other local features such as Gabor filter, etc. But the most important thing is if we can determine whether the compared family is well presented with local or global features. Maybe we can devise a scoring mechanism in a two-phase detection manner, if the sample gets a bad score from the first pass, then the second pass will take place by involving local features. Another experiment we can try is to use Principal Component Analysis (PCA) on the combined features instead of throwing the feature away under the case of UFS or RFE.

Another interesting work we can do is to analyze the malware image, specifically, the executable files. As we discussed in Section 2.2, there are various sections such as .text and .rdata section, etc. In our work we treated those sections evenly, however, we can try to focus more on each section to give them different weights. Furthermore, it would be interesting if we can find out where the most of variations would be spotted, we might be able to target those area(s) for further precise detection process.

Second, in terms of the classification, we can also use a hybrid classifier which might include Naive Bayes, Random Forest, AdaBoost, etc. In that sense, we will be able to get a series of classification results or scores and then we make the final classification decision. For instance, by picking the major vote on the result or classifying with the scores.

Third, from the robust hashing-based perspective, we can try different clustering techniques such as K-means, K-nearest-neighbor, Gaussian Mixture Model, etc. where those techniques are closely related to the concept of robust hashing. From the perspective of the generated secret key, it would be interesting if we view the key as a plain text and the incoming hashed value as a cipher text, then the simple substitution attack might help with solving the clustering problem. And of course, from the side of the compression, it could be replaced by other channel coding algorithms such as Reed-Muller coding or Trellis-Coded Modulation (TCM).

Lastly, we can separate our detection process into several sub-processes. As we suggested before, for SVMs, it performed really good for a small number of families. So we could separate our malware SVM model into different models where each model is customized for different families. The reason is trying to make the SVM training more precise on some specific types of family. Also, different features could be made if there is not only one big malware model is presented.

LIST OF REFERENCES

[1] J. Aycock, *Computer Viruses and Malware*. Springer, 2006.

[2] A. Yewale and M. Singh, "Malware detection based on opcode frequency," in *Advanced Communication Control and Computing Technologies (ICACCCT), 2016 International Conference on*. IEEE, 2016, pp. 646–649.

[3] 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.

[4] R. Venkatesan, S.-M. Koon, M. H. Jakubowski, and P. Moulin, "Robust image hashing," in *Proceedings of 2000 International Conference on Image Processing*, ser. ICIP 2000, vol. 3. IEEE, 2000, pp. 664–666.

[5] G. Conti and S. Bratus, "Voyage of the reverser: A visual study of binary species," *Black Hat*, 2010.

[6] L. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath, "Malware images: Visualization and automatic classification," in *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, ser. VizSec'11. ACM, 2011, pp. 4:1–4:7.

[7] R. Tian, L. M. Batten, and S. Versteeg, "Function length as a tool for malware classification," in *3rd International Conference on Malicious and Unwanted Software*, ser. MALWARE 2008. IEEE, 2008, pp. 69–76.

[8] A. Torralba, K. P. Murphy, W. T. Freeman, and M. A. Rubin, "Context-based vision system for place and object recognition," https://www.cs.ubc.ca/~murphyk/Papers/iccv03.pdf, 2003.

[9] M. Schneider and S.-F. Chang, "A robust content based digital signature for image authentication," in *Proceedings of 1996 International Conference on Image Processing*, ser. ICIP 1996. IEEE, 1996, pp. 227–230.

[10] C.-Y. Lin and S.-F. Chang, "Generating robust digital signature for image/video authentication," in *Multimedia and Security Workshop at ACM Multimedia*, 1998, pp. 49–54.

[11] Y. Zhao, S. Wang, X. Zhang, and H. Yao, "Robust hashing for image authentication using zernike moments and local features," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 1, pp. 55–63, 2013.

[12] J. D. Victor, "Images, statistics, and textures: Implications of triple correlation uniqueness for texture statistics and the julesz conjecture: Comment," *Journal of the Optical Society of America A*, vol. 11, no. 5, pp. 1680–1684, 1994.

[13] V. Monga, A. Banerjee, and B. L. Evans, "A clustering based approach to perceptual image hashing," *IEEE Transactions on Information Forensics and Security*, vol. 1, no. 1, pp. 68–79, 2006.

[14] V. Monga and B. L. Evans, "Robust perceptual image hashing using feature points," in *Proceedings of 2004 International Conference on Image Processing*, ser. ICIP 2004. IEEE, 2004, pp. 677–680.

[15] C.-Y. Lin and S.-F. Chang, "A robust image authentication method distinguishing jpeg compression from malicious manipulation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 2, pp. 153–168, 2001.

[16] M. Johnson and K. Ramchandran, "Dither-based secure image hashing using distributed coding," in *Proceedings of 2003 International Conference on Image Processing*, ser. ICIP 2003. IEEE, 2003, pp. 751–754.

[17] A. Wyner and J. Ziv, "The rate-distortion function for source coding with side information at the decoder," *IEEE Transactions on Information Theory*, vol. 22, no. 1, pp. 1–10, 1976.

[18] G. Conti, S. Bratus, A. Shubina, A. Lichtenberg, R. Ragsdale, R. Perez-Alemany, B. Sangster, and M. Supan, "A visual study of primitive binary fragment types," *Black Hat*, 2010.

[19] M. Stamp, *Introduction to Machine Learning with Applications in Information Security*. Boca Raton: Chapman and Hall/CRC, 2017.

[20] X. Chan and G. Liu, "Discussion of one improved hash algorithm based on MD5 and SHA1," in *Proceedings of the World Congress on Engineering and Computer Science*, ser. WCECS 2007, 2007.

[21] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance," in *International Workshop on Fast Software Encryption*. Springer, 2004, pp. 371–388.

[22] Y. Sutcu, H. T. Sencar, and N. Memon, "A secure biometric authentication scheme based on robust hashing," in *Proceedings of the 7th Workshop on Multimedia and Security*. ACM, 2005, pp. 111–116.

[23] R. W. Hamming, "Error detecting and error correcting codes," *Bell Labs Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[24] S. S. Pradhan and K. Ramchandran, "Distributed source coding using syndromes (DISCUS): Design and construction," *IEEE Transactions on Information Theory*, vol. 49, no. 3, pp. 626–643, 2003.

[25] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 2037–2041, 2006.

[26] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, ser. CVPR 2005. IEEE, 2005, pp. 886–893.

[27] I. Daubechies, "The wavelet transform, time-frequency localization and signal analysis," *IEEE Transactions on Information Theory*, vol. 36, no. 5, pp. 961–1005, 1990.

[28] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.

[29] D. Varodayan, Y.-C. Lin, A. Mavlankar, M. Flierl, and B. Girod, "Wyner-ziv coding of stereo images with unsupervised learning of disparity," in *Proceedings of Picture Coding Symposium*, 2007.

[30] T.-F. Wu, C.-J. Lin, and R. C. Weng, "Probability estimates for multi-class classification by pairwise coupling," *Journal of Machine Learning Research*, vol. 5, pp. 975–1005, August 2004.