2019

Image-based malware classification: A space filling curve approach

Stephen O Shaughnessy

Technological University Dublin, stephen.oshaughnessy@tudublin.ie

Follow this and additional works at: <https://arrow.tudublin.ie/ittscicon>

Part of the [Information Security Commons](#), and the [Other Computer Sciences Commons](#)

Recommended Citation

S. O'Shaughnessy, "Image-based Malware Classification: A Space Filling Curve Approach," 2019 IEEE Symposium on Visualization for Cyber Security (VizSec), 2019, pp. 1-10, doi: 10.1109/VizSec48167.2019.9161583.

This Conference Paper is brought to you for free and open access by the School of Science and Computing at ARROW@TU Dublin. It has been accepted for inclusion in Conference Papers by an authorized administrator of ARROW@TU Dublin. For more information, please contact arrow.admin@tudublin.ie, aisling.coyne@tudublin.ie, gerard.connolly@tudublin.ie.

Image-based Malware Classification: A Space Filling Curve Approach

Stephen O'Shaughnessy*

Technological University Dublin

ABSTRACT

Anti-virus (AV) software is effective at distinguishing between benign and malicious programs yet lack the ability to effectively classify malware into their respective family classes. AV vendors receive considerably large volumes of malicious programs daily and so classification is crucial to quickly identify variants of existing malware that would otherwise have to be manually examined. This paper proposes a novel method of visualizing and classifying malware using Space-Filling Curves (SFC's) in order to improve the limitations of AV tools. The classification models produced were evaluated on previously unseen samples and showed promising results, with precision, recall and accuracy scores of 82%, 80% and 83% respectively. Furthermore, a comparative assessment with previous research and current AV technologies revealed that the method presented here was robust, outperforming most commercial and open-source AV scanner software programs.

Keywords: Space-filling curves, Morton curve, Z-order, malware classification, visualization.

1 INTRODUCTION

The monetization of malware has prompted an upsurge in attacks through schemes such as ransomware, credential stealer Trojans and sextortion scams. AV vendors report receiving upwards of 350,000 samples per day, which makes manual examination impossible and automation essential [1]. Automated tools such as anti-virus scanners and sandboxes can quickly distinguish between benign and malicious programs, but for the most part, do not classify malware into their correct taxonomic family classes. Classification of malware is crucial to the automation process, since it can determine if a previously unseen malicious program is a new or zero-day attack or simply a variant of an existing malware. This will greatly reduce the level of manual intervention required for variant samples, leaving analysts more time to focus on new malware strains. Clearly, there is a need for a solution to the classification limitations of AV scanners that is also scalable to the large volumes of samples received by AV vendors daily.

Malware classification, like malware analysis, can be broadly divided into two distinct areas: static and dynamic. Static analysis involves inspection of malware code and structures in a non-running state, whereas dynamic analysis comprises monitoring the malware in execution and examining behavioural data. A critical step in the classification process is feature extraction, which creates a subset of characteristic and discriminant features from a vague collection of data. Feature extraction can also be carried out through static or dynamic means.

Static features are extracted from non-running code by calculating frequency or sequence metrics, such as n-gram or PE file header analyses [2][3]. Dynamic features are extracted by executing malware and recording the behaviour by monitoring its interactions with the operating system, for example through API call sequences and graph representations [4][5]. However, these methods are not without their drawbacks. For example, n-gram analysis can only capture short-distance context dependency within the n-words window, so any word contexts outside of the n range are not recognised; PE header analysis and dynamic methods can produce a high volume of false positives as similar features may be present in multiple malware families or even benign files.

Visualization and computer vision techniques have been explored as an alternative to the more typical methods for malware classification [6][7][8]. Visualisation is an effective way to represent data, as it can summarize large volumes of binary or textual data into a single source that can be interpreted by the human eye or form a basis for further image processing. Malware classification in this case consists of mapping the binary malware files to 2-dimensional images. The resulting images contain a series of characteristic textures that can be extracted as discriminant features to create 'fingerprints' for identification and classification. This paper presents a novel method of image-based malware classification using space-filling-curves. Space-filling curves are a family of mathematical functions that consist of a continuous curve which passes through every point of a regular spatial region, such that the spatial locality of the data is preserved. Therefore, data that are close together in a one-dimensional binary will be grouped close together in the two-dimensional visual space. The properties of space-filling curves allow malware variants containing similar code to map to images with similar patterns, from which discriminant texture features may be extracted for classification.

The main contributions of this research are:

- An evaluation of Space-filling curves for use in image-based malware classification.
- A scalable solution for classification ambiguity among AV programs.
- Three open source SFC datasets made available for further research [9].

*stephen.oshaughnessy@tudublin.ie

2 RELATED RESEARCH

2.1 Image-based Malware Classification

Previous research has focused predominantly on the byte-to-pixel representation for image-based malware classification, where each byte in the binary file is directly mapped to a corresponding pixel. This mapping, referred to as a “byteplot”, was first introduced by Conti et al. as a method of reverse engineering binary files into a visual image to enhance the capabilities of text hex editors [10][11]. Nataraj et al. were the first to apply the byteplot mapping to malware classification [12]. The authors derived feature vectors using GIST, a global feature descriptor that breaks images into sub-band blocks and computes features based on filters tuned to varying scales and orientations. For evaluation, the authors compiled a dataset, dubbed Maling, comprising 9,548 samples from 25 malware families [13]. In [14] Nataraj et al. evaluated the byteplot approach against dynamic analysis methods. Their findings showed that, while binary-texture analysis provided comparable classification accuracy results to that of contemporary dynamic methods, processing time was 4000 times faster, with average processing times of 60MS for each image compared to 4 minutes for dynamic feature extraction. The work presented by Nataraj et al. can be considered a benchmark in this field as several subsequent research efforts have extended or been compared to their work. In [15], Luo and Lo extracted features for classification using TensorFlow and Local Binary Patterns (LBP) feature descriptors on the Maling dataset. The authors reported a classification accuracy of 93.17% as opposed to 82.83% using the method proposed by Nataraj et al. In [16], Yajamanam et al. conducted experiments to compare the GIST-based method using a subset of features taken from the original method, which increased processing time to 20ms per sample and returned a classification accuracy of 98%. [17] Naeem and Naeem used a combination of local and global feature descriptors to classify the Maling dataset, reporting a classification rate of 97.4%.

In contrast, little research has been conducted on space-filling curves in this context. In [18], Baptista presented research in classifying malware by type using Hilbert curves and a Self-Organizing Incremental Neural Network. Due to the limited size of the dataset, comprising 180 samples, 78 of which were benign, this research would be considered insufficient to give any clear indication of the merits of applying the Hilbert curve to malware classification. Irwin and Pilkington [19] applied the Hilbert space-filling curve to the problem of mapping malicious network traffic, specifically the Blaster Internet worm. This mapping was for visualisation purposes and performed no classification of the malware samples.

2.2 Space-filling Curves

Space filling curves (SFC’s) are mathematical constructs, also known as continuous fractal curves, the limit of which contain the entire 2-dimensional unit square. First introduced by Peano in 1890 [20], there are several implementations, based on differing traversal orders. In 1891, Hilbert defined a simplified variant of the Peano curve [21]. Other variations include Moore’s curve [22], Sierpiński’s curve [23], Gray-code curve [24] and Z-order or Morton curve [25]. The Hilbert, Z-order and Gray-code curves are evaluated in this research. In simple terms, a space-filling curve follows a continuous path until it fills an n-dimensional space entirely. SFCs can extrapolate data from one dimension into n-dimensions while preserving the properties of the original data, with the property that closely related data are grouped together in the resulting n-dimensional space.

In [26], Wattenberg described a connection between space-filling visualisations and the mathematics of space-filling curves. In doing so, the author derived a proof for a perfect layout function using jigsaw tree-maps and space-filling curves, identifying four desirable properties of a layout function:

- *Stability*: The ordering of information must be preserved.
- *Split neutrality*: structural changes are handled smoothly so as other regions are not affected.
- *Order adjacency*: each similar item should be located adjacent to each other.
- *Locality*: layout regions should be relatively compact, rather than long and narrow, to aid in clearly visualising regions.

Figure 2.1 presents a schematic view of how the space-filling curve can be used to map data to an image. At left is a one-dimensional layout corresponding to the binary data; the centre is the same layout extended to a screen-filling curve, and at right is the actual space-filling result. In this case, the different colour codes represent different data types, such as printable characters or null bytes.

Figure 2.1: Mapping one-dimensional data to a two-dimensional SFC image. (image courtesy of Wattenberg [26])

Binvis is an online interactive binary visualization tool that generates images from binary files using space-filling curves [27]. Binvis is implemented using scurve, a library written in Python that visualises binary files in several different curve representations [28]. The scurve library was used in this research to generate the images for classification. The scurve library uses four different colour codes to map binary files using SFCs: white for 0xFF values, black for 0x00, blue for printable characters and red for unreadable or high entropy data. The latter includes encrypted, encoded or compressed data so images displaying significant areas of red signify malware that has been obfuscated in some way.

3 METHODOLOGY

The research presented in this paper provides a study of the efficacy of space-filling curves as a means of representing malware as 2-dimensional images that can be used to classify malware through computer vision and machine learning. The method of the study comprises four distinct parts: data gathering, data conversion, feature extraction and classification.

3.1 Data Gathering

For the purposes of the evaluation experiments, a malware dataset was compiled from the current VirusTotal academic collection, using a selection of malware samples captured since January 2018. The dataset comprised 9,235 Windows 32-bit executable samples from 28 distinct families. The dataset represents a more up-to-date set of samples than the Maling and Kaggle [29] datasets, with some of the most prevalent and current malware species in circulation today, including many examples of obfuscated malware. In an effort to reduce the possibility of mislabeling, each sample was

compared for similarity using the ssdeep similarity hashing algorithm [30]. ssdeep is an approximate-matching function that compares files for similarity, using a scoring system from 0 for no similarity to 100 for a high degree of similarity. For this research, only samples scoring a threshold of 60 or above were retained, such that each sample in the data set had at least a score of 60 compared with every other sample in that family class. Since malware variants share many similar characteristics, it was assumed that files above this threshold were from the same family of malware. After filtering, the resulting dataset contained 8,452 samples. Appendix I shows the structure of the dataset, including the family type and count for each class. It should be noted that benign files were not included in the dataset, as the aim of this research was to classify variants into their correct family classes, rather than detect between malicious and benign samples.

3.2 Data Conversion

In mathematical analysis, a space-filling curve is a curve whose range contains the entire 2-dimensional unit square or more generally an n -dimensional unit hypercube, but for the purposes of this research only the 2-dimensional space is considered, since the output is a 2-dimensional image. In this case, the 2-dimensional unit square represents an image of $n \times n$ pixels. SFC's trace a continuous curve through every unit square, i.e. pixel in the image. To envision this for the purpose of malware binary-to-image mapping, the code of the program can be considered a flattened out 1-dimensional line with the code being parsed sequentially, byte by byte. The SFC maps each point (bytes) from the 1-dimensional space to the 2-dimensional space (SFC image map) such that closely located points in the binary file space will tend to also be closely located when mapped using the SFC. Figure 3.1 illustrates the traversal patterns of the Z-order, Gray-code and Hilbert curves onto the 2-dimensional space.

Figure 3.1: (a) Z-order, (b) Gray-code, (c) Hilbert curve map ordering onto a 2-dimensional space

Following the conversion process using the scurve library, three datasets were produced, one for each SFC mapping. The Hilbert conversion method first determines the dimensions of input binary. It then iterates through each coordinate point and maps this to its data color class as discussed previously. The implementation of the Z-order and Gray-code curves are similar, in that the scurve library considers the bit range for each distinct chunk of binary data as coordinates. In other words, the start offset is the first coordinate and the end offset is the second coordinate. In the case of the Z-order code, a bit interleaving process then takes place. First, decimal coordinates are converted to binary. Next, a corresponding bit is taken from each binary coordinate and concatenated. This is done in a cyclical fashion, resulting in a single number, called a Morton code. For the Gray-code, the coordinates are XORed from left to right. The resulting codes in each case represent the cell that the binary chunk will occupy, which is mapped to a distinct colour, depending on the data type.

Figure 3.2 shows the same variant sample taken from the Allapple worm family, transformed into 2-dimensional tree-map images using the Hilbert, Gray-code and Z-order formats. While similarities can be drawn in terms of the texture regions within the resulting images, it can be seen the layout patterns are distinctly different.

Figure 3.2: scurve SFC representations of the same Allapple worm sample - (a) Hilbert, (b) Gray-code and (c) Z-order curves

3.3 Feature Extraction

The feature extraction process is carried out using image feature descriptors, which represent a set of salient points that can be extracted from an image. These points can then be used as discriminant characteristics to classify the image. Three feature descriptors were chosen for evaluation based on their suitability for texture feature extraction, namely Local Binary Patterns (LBP), Gabor filters and Histogram of Gradients (HOG). The basic LBP operator is a simple yet efficient texture descriptor, first introduced by Ojala [31] which labels pixels of an image by thresholding a 3×3 neighborhood of each central pixel and converting the result into a binary number. The resulting LBP code represents the grayscale intensity for that center pixel. This research used an extension of the original LBP, proposed by Pietikäinen [32], which accounts for variable neighborhood sizes, making it more effective at varying scales. Gabor filters [33] are linear filter for texture analysis. Gabor filters have been shown to possess optimal localization properties in both spatial and frequency domain and thus are well suited for texture segmentation problems. They work by analyzing an image in a specified orientation (direction), spatial width and frequency. The HOG descriptor was first introduced by McConnell in a patent for a pattern recognition method [34]. The HOG descriptor is primarily used for object detection in images but has produced favorable results in texture analysis, e.g., [35][36]. HOG descriptors work by describing the distribution of intensity gradients or edge directions in a localized area of an image. A custom script in Python was used to determine the optimum parameters for each feature descriptor algorithm.

3.4 Classification

Supervised learning algorithms were used exclusively in this research since the class labels, i.e., the malware family names, were already known. Supervised learning maps an inferred function between the training data and the corresponding class label and uses this function to identify class labels for unseen samples. Three supervised machine learning algorithms were trained on the SFC datasets: K-Nearest Neighbor (KNN), Random Forest (RF), and Decision Trees (DT). Naïve Bayes and Support Vector Machines were also tested but provided poor results and so are not presented here. A crucial step in the model training phase is the choice of parameters for the classifier. A search method was implemented to determine the optimal combination of hyper-parameters, namely GridSearchCV, which is part of the Python Sci-kit Learn library [37]. With GridSearchCV, all the possible combinations of

parameter values are evaluated, and the best combination is retained.

3.5 Precision Metrics

In order to ensure robust classification models, their performance must be tested. In this research, the performance metrics precision, recall accuracy were chosen. Confusion matrices provided a graphical view of how well each model performed. The performance metrics precision, recall and accuracy are calculated using values derived from the outcome of classification process. These are:

- *True positive (TP)*: all instance of a class that are classified as that class.
- *True negative (TN)*: all non-instances of a class that are not classified as that class
- *False positive (FP)*: all non-instances of a class that are incorrectly classified as that class
- *False negative (FN)*: all instances of a class that are not classified as that class

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. In this case, the ratio of correctly predicted samples for a family to the total predicted as that family i.e., ‘for all the malware labelled as a particular family, how many were correct?’ The formula for calculating precision is given in (1):

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

Recall is the ratio of correctly predicted positive observations to the all observations in actual class. In this case, it is the ratio of the correctly predicted malware to the total number for that family, i.e., for each malware family, how many that should have been labelled as that family, were labelled correctly? The formula for recall is given in (2):

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

Accuracy is the fraction of correct predictions that the model predicted correctly. The formula is given in (3):

$$Accuracy = \frac{TP}{TP + FP + TN + FN} \quad (3)$$

Confusion matrices are a graphical way of illustrating the performance of a classification model. A confusion matrix will show true positives, true negatives, false positives and false negatives; hence they can be used to calculate precision, recall and accuracy metrics. The matrix rows represent the actual values of the model, whereas the columns represent the predicted values. The intersection of row and column give the true positive rate, i.e. the model predicted the actual values correctly.

3.6 Validation

Efforts were made to reduce underfitting overfitting in the classification models. Underfitting can occur when there is not enough data to build a generalized model. Ultimately, the model will fail to identify signatures or patterns in the data. Overfitting occurs when the model fits the data too well, mainly due to it

capturing noise along with the underlying signatures or patterns in the data. A model that is trained to fit slightly inaccurate data can infect it with substantial errors and reduce its predictive power. To reduce the risk of underfitting, a suitably large dataset was compiled. Stratified k-fold cross-validation was used to minimize overfitting. Stratification is the process of dividing members of the data population into homogeneous subgroups before sampling. In stratified k-fold cross-validation, folds are stratified such that they contain approximately the same proportion of samples as the original dataset. Cross-validation helps identify if overfitting is present in the data by repeating the training and testing phases multiple times, using all the different folds of the training set as validation sets. Through experiments, the optimal value for k was found to be 5.

4 EXPERIMENTAL ANALYSIS

Experimental analysis was carried out using the three SFC image datasets described in section 3.2. Approximately 10% of the data in each set was held for evaluating the classification models. Each classification model was trained following the same procedure:

1. Compute the feature vector for each image sample.
2. Assign the family label to each vector.
3. Pass the feature set (vectors and labels) to a classifier.
4. Perform k-fold cross-validation.
5. Compute the average precision, recall and accuracy.
6. Compute the confusion matrix.

The performance results from the training phase are given in Appendix II. Out of all the models tested, the KNN-HOG model performed best using the Z-order dataset with precision, recall and accuracy scores of 94.5%, 87.1% and 91.6% respectively.

The KNN-HOG Z-order model is now considered for closer examination. The most important parameters to consider when tuning the HOG descriptor are orientation, pixels-per-cell and cells-per-block. The orientation parameter represents the number of orientation bins in the resulting histograms. Pixels-per-cell determine the size, in pixels, of each cell. Cells-per-block determine the size of each block in pixels. The optimum parameter values determined were *orientation* = 16, *pixels-per-cell* = 60 and *cells-per-block* = 1. *L2-Hys normalisation* was used in order to help improve invariance to changes in illumination. For KNN classifier, the most important metrics are the number of neighbours, *k*, and the *distance metric* used to calculate the distance between data points or neighbours. The optimum parameter values returned were *k*=1 and the ‘cityblock’ distance measure.

Figure 4.1 shows the normalised confusion matrix for the KNN-HOG model. The true positives are shown along the diagonal, on a scale of 0 to 1, where 0 indicates no samples were predicted correctly and 1 indicates all samples were predicted correctly. The matrix shows that the model performed well in the training and testing phase, with a true positive rate of 0.8 or above in 23 out of 28 families. The remaining families require further examination, as their performance was below average. These were: Graftor (0.78), Chir (0.70), Cryptowall (0.73), Locky (0.77) and Bitman (0.78) families.

For each of these families, it was observed that most variants were obfuscated in some way. Obfuscation, in this case, was ascertained by calculating the Shannon entropy for each variant. Shannon [38]

Figure 4.1: KNN-HOG model confusion matrix

introduced a proof and formula for calculating entropy, which is a measure of the degree of randomness for a given set of data. Shannon entropy ranges from 0 for orderly or non-random data, to 8 for very random data. Obfuscated data exhibits a high degree of randomness and so tends towards a high entropy score. Generally packed malware will score between 5.5 and 6.5, whereas encrypted malware will score above 6.5. The average entropy score for each of the lowest scoring families is shown in Table 4.1.

Table 4.1: Obfuscated malware entropy scores

Family	Entropy
Locky	7.63
Chir	5.76
Cryptowall	7.51
Bitman	6.90
Graftor	6.99

It is evident from the high entropy scores in Table 4.1 that the samples tested were obfuscated in some way, which impacted the performance of the classification model for those families. The confusion matrix also indicated a high correlation between the Bitman and TeslaCrypt families, where the model predicted 21% of Bitman samples as TeslaCrypt. Figure 4.2 illustrates samples taken from both families. Visually, there are significant areas of similar texture shared between the two sample sets, which attributed to the model confusing TeslaCrypt for Bitman. Both family sample sets were inspected, and visual similarities were observed in a high percentage of cases. To support the visual evidence, a deeper examination was carried out by disassembling the malware binaries and comparing the code. It was discovered that considerable chunks of code were very similar or identical. Both families are types of ransomware malware, which would explain some resemblances in code due to similar functionality.

There is also a possibility of the malware being mis-labelled during the data compilation phase. While precautions were taken to prevent mis-labelling malware samples, as described in section 3.1, some mislabelling may still have occurred. Additionally, if similar obfuscation tools are used for encryption or packing, there may be similarities between variants caused by signatures generated by the tool. Ultimately, the solution to these limitations would be to manually analyse all samples for positive identification, which would not be feasible at scale.

Figure 4.2: (a) Bitman and (b) TeslaCrypt SFC samples

4.1 Model Evaluation

The KNN-HOG model was evaluated on the 10% subset, removed before the model training phase. Stratified random sampling was used to select the evaluation set in order to provide a representative of the original sample data. Appendix III shows the performance results from evaluating the classification models on the three SFC datasets. As in the training phase, the KNN-HOG model performed best on the unseen data, with precision, recall and accuracy scores of 82%, 80% and 83% respectively. The performance metrics show that the KNN-Model model trained on the Z-order data generalises well to new data, as it can identify variants it has not seen previously.

4.2 Time and Space Complexity

The time complexity of both the image processing and classification stages were measured to evaluate the scalability of the method presented here. If the method is to be considered a feasible solution to the classification limitations of AV software, it must be scalable to large volumes of data. The average time taken to extract HOG features from each image was 6.88ms. The classification time per image was 1ms. The time complexity of both the image processing and classification stages can be considered $O(n)$ or linear as they will both grow proportionately to the size of the data. The fast processing times per sample, coupled with $O(n)$ time complexity indicate that this method would be scalable to massive datasets, such as the malware data processed by AV vendors daily. In terms of space complexity, the processed images require considerably less disk space than the original executables. The Z-order image dataset is 183MB in size, with an average size of 24KB per image, whereas the original executable dataset is 5.2GB with an average of 681KB per sample.

5 COMPARATIVE ANALYSIS

To test the model in a more real-world setting, two comparative scenarios were set up. The first was a comparison with the byteplot method presented by Nataraj et al. in [11], described previously. The second scenario compared the KNN-HOG model against the 70 AV scanners on VirusTotal.

5.1 Comparison with GIST Byteplot Method

The KNN-HOG model was tested against the GIST method proposed by Nataraj et al. This method uses byteplot mapping to represent malware binaries as images. Feature extraction is performed using GIST descriptors, which are global image feature descriptors that break images into sub-band blocks and compute features based on filters tuned to varying scales and orientations. KNN is used as the classifier. Both methods were trained and evaluated in the same manner as discussed in section 4. Table 5.1 provides a comparison of results for both methods.

Table 5.1: GIST vs. KNN-HOG performance

Method	Training			Evaluation		
	Prec.	Rec.	Acc.	Prec.	Rec.	Acc.
GIST	91.0	91.0	91.0	77.2	73.4	78.3
KNN-HOG	91.3	91	91.3	82.4	80.6	83.0

Prec = Precision; Rec = Recall; Acc = Accuracy

From table 5.1, both methods performed comparatively well in the training phase with precision, recall and accuracy scores of approximately 91% for each model. However, the KNN-HOG method outperformed the GIST method on the evaluation dataset, meaning that it generalized better to previously unseen data samples.

To give a further comparison of the models' prediction accuracies, the following metrics were considered:

- *True positive rate (TPR)*: The proportion of positive cases identified correctly
- *True negative rate (TNR)*: The proportion of negative cases identified correctly
- *False positive rate (FPR)*: The proportion of negative cases incorrectly identified as positive cases
- *False negative rate (FNR)*: The proportion of positive cases incorrectly identified as negative cases

In the case of multi-class classification, a positive case is when a sample is belonging to a particular class (malware family), whereas a negative case is when the sample belongs to a different class. The rate metrics are measured between 0 and 1. For the true rates, a rate of 1 is most accurate. For the false rates, a value of 0 is most accurate. Table 5.2 gives a summary of the results for the two models.

Table 5.2: Model accuracy rates

Rate	KNN-HOG	GIST
TPR	0.94	0.88
TNR	0.99	0.99
FPR	0.002	0.004
FNR	0.06	0.12

While both models returned high true rates, the KNN-HOG model returned a higher TPR rate, meaning it identified more true cases correctly. Both models had low FPR rates, which means that for each sample they did not predict many as belonging to another family. The FNR was higher for the GIST model, at 12% compared to the KNN-HOG model at 6%, meaning that, for each sample the KNN-HOG model predicted less samples incorrectly as other families. To give an example of this, out of a possible 55 Teslacrypt samples, the GIST model correctly identified 44 samples correctly, but incorrectly identified 10 samples as Bitman and 1 as Locky. In contrast, the KNN-HOG model identified 53 Teslacrypt samples correctly, incorrectly identifying 1 as Zmist and 1 as Bitman. The results of the tests show that the KNN-HOG is a more robust model, identifying less samples incorrectly.

5.2 Comparison with AV Scanners

VirusTotal is an online scanning engine, comprising 70 individual AV scanners and provides reports on suspect files, URL's and cryptographic hashes [39]. The aim of this comparative assessment was to establish how well the KNN-HOG model compared against multiple commercial and open source AV scanning programs. The samples from the previous experiments were used for testing. The samples were uploaded to VirusTotal via a Python script through the integrated VirusTotal API [40]. For each family, the resulting scan reports were examined for the family variant name. A sample was only considered as classified correctly if the AV scanner result contained the variant name. The bar graph in Figure 5.1 represents the classification rates of the KNN-HOG model (blue) and VirusTotal scanners (red). It is clear from the graph the KNN-HOG model outperformed the aggregated group of AV scanners by a considerable margin on all samples tested, with the exception of the Win32.Neshta family. In this case, the classification accuracy for KNN-HOG model was 14% compared with 25% for the AV scanners. Overall, the average classification rate for the KNN-HOG model was 82.1%, compared to 12.7% for the VirusTotal scanners.

It should be noted that detection rates for the AV scanners were considerably higher, with an average detection rate of 77.1%. For example, although the classification rate for Dridex was 1.5%, the detection rate was 83.37%. Upon inspection, the majority of detections were assigned generic labels, such as *Trojan.Downloader* or *malicious_confidence_100%*. In these cases, the AV scanners had detected malicious signatures in the samples but failed to classify the malware into its correct family class and so the sample is given a more generalized label. This experiment highlights the ambiguity problem that is prevalent in AV scanning tools.

Efforts have been made to reduce classification ambiguity by standardizing the naming of malware, such as the Computer Antivirus Research Organization (CARO) and the Malware Attribute Enumeration and Characterization (MAEC). CARO introduced a structured naming convention for malware in 1991 [41]. In the CARO scheme, malware is identified by type, target platform, family and variant, e.g. *Trojan.Win32.Bitman.A*. Whilst this method of naming is intuitive, few AV companies have adopted this scheme fully and of those that do, many opt to use a customized variation, which can add to the naming disparity between vendors. MAEC is a structured language for encoding and sharing information about malware, based upon attributes such as behaviors, artifacts, and relationships between malware samples [42]. The aim of MAEC is to provide a portable language containing highly granular descriptions of malware with the aim of reducing ambiguity between malware families. In a similar vein, MAEC has yet to be adopted by most AV vendors. Without

Figure 5.1: KNN-HOG Vs. VirusTotal scanners classification accuracy

widespread opt-in from AV vendors, this ambiguity problem will worsen as numbers of malware increase, highlighting need for a robust, scalable classification solution.

6 CONCLUSIONS AND FURTHER RESEARCH

The findings of the experimental and comparative analyses in this paper reveal the potential use of space-filling curves for image-based malware classification. Through experiments, it was shown that malware variants can be mapped to a 2-dimensional space using space-filling curves, from which discriminant features can be extracted. Using computer vision and machine learning methods, the features can be used to classify malware variants into their respective families with a high degree of accuracy. The proposed model was compared to the GIST byteplot method by Nataraj et al and performed favorably, particularly against unseen data. Results from comparative analyses with VirusTotal show the KNN-HOG model can outperform the majority of commercial and open-source AV scanner software programs with respect to classification accuracy. The methods produced by this research could be used to augment the performance of automated malware analysis tools by increasing the classification rates of variants and thereby reducing the ambiguity in malware labeling. $O(n)$ processing time of the proposed method highlights its potential to work on large scale datasets. It is also modular, in that that it can be adapted easily to include new families or variants.

Current research by the author has focused on using dynamic analysis through virtual machine introspection (VMI) in conjunction with process memory dump parsing to extract dynamic features from heavily obfuscated malware. As shown through experimental analysis in Section 4, obfuscation presents an issue for static analysis, since the data is scrambled by the encryption or packing routines it is obfuscated with, making extraction of discriminant features difficult. Dynamic analysis is immune to obfuscation, since malware must be decrypted or de-compressed when written to system memory before execution. In this case a suspect binary is executed within a virtual machine and a memory dump of the malicious process is produced, which is then mapped

to an SFC image, as described previously. The resulting image represents a visualisation of the behavior data, i.e., the process information of the malware that is resident in memory.

dump images for family variants are then classified in the same way as the methods described in this paper. Initial testing on both obfuscated and un-obfuscated malware samples has proved to be promising, with considerable improvements in classification performance over the static method alone.

A potential avenue of further research would be to implement image segmentation for classification at a more granular level. Segmentation could be used to partition the images into distinct regions containing groups of pixels with similar textures. Regions of interest in an executable binary could be extracted in this way, leaving irrelevant data such as padding (regions of binary zeros) or obfuscated sections, where the entropy would otherwise adversely affect the classification algorithms.

7 REFERENCES

- [1] Kaspersky lab (2018). Kaspersky Security Bulletin: Overall Statistics for 2017. [online] Available at: https://media.kaspersky.com/jp/pdf/pr/Kaspersky_KSB2017_Statistics-PR-1045.pdf. Accessed: 13/10/2018.
- [2] M.Z. Shafiq, S.A. Khayam, M. Farooq (2008) Embedded Malware Detection Using Markov n-Grams. In: Zamboni D. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2008. Lecture Notes in Computer Science, vol 5137. Springer, Berlin, Heidelberg.
- [3] T. Wang, C. Wu, and C.Hsieh (2009). Detecting unknown malicious executables using portable executable headers. In INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on, pages 278–284, Aug 2009.
- [4] S. Gupta, H. Sharma, S. Kaur (2016) Malware Characterization Using Windows API Call Sequences. In: Carlet C., Hasan M., Saraswat V. (eds) Security, Privacy, and Applied Cryptography Engineering.

- SPACE 2016. Lecture Notes in Computer Science, vol 10076. Springer, Cham.
- [5] K.A. Roundy, B.P. Miller (2010) Hybrid Analysis and Control of Malware. In: Jha S., Sommer R., Kreibich C. (eds) Recent Advances in Intrusion Detection. RAID 2010. Lecture Notes in Computer Science, vol 6307. Springer, Berlin, Heidelberg.
- [6] M. Wagner, F. Fischer, R. Luh, A. Haberson, A. Rind, D. A. Keim and W. Aigner (2015). A Survey of Visualization Systems for Malware Analysis. Proceedings of: Eurographics Conference on Visualization (EuroVis), 2015.
- [7] H. Inoue, F. Adelstein, R. Joyce (2011). Visualization in Testing a Volatile Memory Forensic Tool. The Digital Forensic Research Workshop (DFRWS 2011) USA. New Orleans, LA (Aug 1st - 3rd 2011).
- [8] K. Kancherla und S. Mukkamala (2010). Image visualization-based malware detection. In 2013 IEEE Symp. On Computational Intelligence in Cyber Security (CICS), 2013, p. 40–44
- [9] SFC datasets for research. Available online at: <https://github.com/stephen-oshaughnessy/SFC-datasets>.
- [10] G. Conti, S. Bratus, A. Shubina, A. Lichtenberg, R. Ragsdale, R. Perez-Aleman, B. Sangster, and M. Supan (2010). A Visual Study of Binary Fragment Types. Black Hat USA.
- [11] G. Conti, E. Dean, M. Sinda and B. Sangster (2008) Visual reverse engineering of binary and data files. In Visualization for Computer Security, Proc. VizSec (2008), LNCS 5210, Springer, pp. 1–17.
- [12] L. Nataraj, S. Karthikeyan, G. Jacob, B.S. Manjunath (2011) Malware images: Visualization and automatic classification. In Proc. 8th Int. Symp. Visualization for Cyber Security, VizSec (2011), ACM, pp. 4:1–4:7.
- [13] Maling (Nataraj et al.) (2011). Maling dataset. [online] Available at: https://www.dropbox.com/s/ep8qjakfwh1rzk4/8icross_dataset.zip?dl=0. Accessed: 17/2/2019.
- [14] L. Nataraj, Y. Vinod, P. Porras, and J. Zhang (2011). A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In Proceedings of the 4th ACM workshop on Security and artificial intelligence (AISec '11). ACM, New York, NY, USA, 21-30.
- [15] J. Luo and D. C. Lo (2017). Binary malware image classification using machine learning with local binary pattern. 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, 2017, pp. 4664–4667.
- [16] S. Yajamanam, V. Raja, S. Selvin, F. Di Troia, M. Stamp (2018). Deep Learning versus Gist Descriptors for Image-based Malware Classification. Proceedings of the 2nd International Workshop on FORmal methods for Security Engineering, January 2018.
- [17] H. Naeem and M.R. Naeem (2018). A light-weight malware static visual analysis for IoT infrastructure. 2018 International Conference on Artificial Intelligence and Big Data (ICAIBD). 26th-28th May 2018, Chengdu, China.
- [18] I. Baptista (2018). Binary visualisation for malware detection. The Plymouth Student Scientist, 2018, 11 (1), 223-237.
- [19] B. Irwin and N. Pilkington (2007) High Level Internet Scale Traffic Visualization Using Hilbert Curve Mapping. VizSEC 2007, Proceedings of the Workshop on Visualization for Computer Security, Sacramento, California, USA, October 29, 2007.
- [20] G. Peano (1890). Sur une courbe, qui remplit toute une aire plane, Mathematische Annalen (in French)
- [21] D. Hilbert (1891). About the continuous image of a line on a surface piece. Mathematische Annalen/Zeitschriftenband (1891)/ Periodicals / Articles / 459 - 460 (in German).
- [22] E.H. Moore (1900) On Certain Crinkly Curves, Trans. Amer. Math Soc., 1, 72-90 (1900).
- [23] W. Sierpiński (1915) "Sur une courbe cantorienne dont tout point est un point de ramification" C.R. Acad. Sci. Paris , 160 (1915) pp. 302.
- [24] F. Gray (1953), Pulse code communication (NB. U.S. Patent 2,632,058 filed November 1947).
- [25] G.M. Morton (1966), A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing (PDF), Technical Report, Ottawa, Canada: IBM Ltd.
- [26] M. Wattenberg (2005). A note on space-filling visualizations and space-filling curves. IEEE Symposium on Information Visualization, 2005. INFOVIS 2005., Minneapolis, MN, 2005, pp. 181-186.
- [27] A. Cortesi (2018) Binvis binary visualisation tool. [online] Available at: <http://binvis.io/>. Accessed: 24/9/2018
- [28] A. Cortesi (2018) Scurve space-filling curve library in Python. [online] Available at: <https://github.com/cortesi/scurve>. Accessed: 24/9/2018.
- [29] Kaggle (2015). Microsoft Malware Prediction Competition Dataset. [Online] Available at: <https://www.kaggle.com/c/microsoft-malware-prediction/data> Last accessed: 18/11/2018.
- [30] J. Kornblum (2006). Identifying almost identical files using context triggered piecewise hashing Journal of digital investigation 3S (2006) S91–S97.
- [31] T. Ojala, M. Pietikainen and D. Harwood (1996). A comparative study of texture measures with classification based on feature distributions. Pattern Recognition 29(1), 51–59 (1996).
- [32] T. Ojala, M. Pietikäinen and T. Mäenpää (2000). Gray Scale and Rotation Invariant Texture Classification with Local Binary Patterns. In: Computer Vision - ECCV 2000. ECCV 2000. Lecture Notes in Computer Science, vol 1842. Springer, Berlin, Heidelberg.
- [33] D. Gabor (1946) Theory of Communication. Journal of the Institution of Electrical Engineers, 93, 429-441.
- [34] R.K. McConnell (1986) Method of and apparatus for pattern recognition. U.S. Patent No.4, 567,610, January 1986.
- [35] H.M. Harb, A.S. Desuky and R. Jennane (2017). Histogram of Oriented Gradients and Texture Features for Bone Texture Characterization. International Journal of Computer Applications 165(3):23-28.
- [36] H. Demir (2018). Classification of Texture Images Based on the Histogram of Oriented Gradients Using Support Vector Machines. Istanbul University - Journal of Electrical and Electronics Engineering 18(1):90-94.
- [37] Sci-kit (2019). Sci-kit Machine Learning Library for Python. [Online] Accessed at: <https://scikit-learn.org/stable/>. Last accessed: 4/2/2019.
- [38] C. E. Shannon (1948). A Mathematical Theory of Communication. Bell System Technical Journal. 27 (3): 379–423.
- [39] VirusTotal (2019). VirusTotal aggregated malware scanner. [Online] Accessed at: <https://www.virustotal.com/reference>. Last accessed: 21/2/2019.
- [40] VirusTotal API (2019). VirusTotal API reference. [Online] Accessed at: <https://developers.virustotal.com/reference>. Last accessed: 21/2/2019.
- [41] CARO (2019). Computer Antivirus Research Organization. [Online] Accessed at: <http://www.caro.org>. Last accessed: 4/8/2019
- [42] MAEC (2019). Malware Attribute Enumeration and Characterization. [Online] accessed at: <https://maecproject.github.io/>. Last accessed: 4/8/2019.

8 APPENDIX I: MALWARE DATASET STRUCTURE

Malware Family	Type	Count
Teslacrypt	Ransomware	1257
W32.Wabot	Worm	688
Installcore	Bundle installer	631
W32.Wapomi	Virus	580
Bitman	Trojan	518
Vilse	Trojan	439
Cerber	Ransomware	386
W32.Virut	Virus	381
Emotet	Banker Trojan	363
Win32.Allaple	Worm	362
Win32.Dinwod	Trojan Dropper	343
Zusy	Spyware	300
Cryptex	Virus	299
Bundl	PUA	263
InstallMonster	Adware Installer	250
Shifu	Banker Trojan	241
Androm	Backdoor Bot	166
Autoit	Compiler	137
Zmist	Polymorphic Virus	128
Cosmu	Trojan	105
Win32.Neshta	Virus	93
Locky	Ransomware	93
Cryptowall	Ransomware	93
Win32.Scar	Trojan	87
Dridex	Banker Trojan	70
Graftor	Adware	65
Trojan.Agent.BDMJ	Trojan Adware	63
Win32.Chir	Email Worm	49

9 APPENDIX II: MODEL TRAINING PERFORMANCE METRICS

Model	Hilbert			Z-order			Gray-code		
	Prec.	Recall	Acc.	Prec.	Recall	Acc.	Prec.	Recall	Acc.
KNN-LBP	89.9	87.1	89.7	90.0	88.8	90	88.2	87.6	90
RF-LBP	92.3	85.8	89.9	92.8	87.7	90.1	91.5	86.3	90
DT-LBP	82.6	80.8	85.3	84.7	83.4	85.1	84.3	82.5	85.1
KNN-Gabor	89.9	87.4	90	89.1	88.6	89.2	88.1	87.1	89.2
RF-Gabor	93.1	85.3	89.9	92.8	85.4	90.2	92.2	85.9	89.8
DT-Gabor	83.2	82.5	84.6	82.5	82.1	85.4	84.3	83.4	86
KNN-HOG	94.0	85.3	91.6	94.5	87.1	91.6	94.0	85.1	91.6
RF-HOG	92.1	90.9	89.1	92.3	91.4	89.5	92.6	91.1	89.5
DT-HOG	79.3	78.0	81.3	82.0	81.0	82.3	80.3	79.0	82.4

Prec. = Precision; Acc. = Accuracy

10 APPENDIX III: MODEL EVALUATION PERFORMANCE METRICS

Model	Hilbert			Z-order			Gray-code		
	Prec.	Recall	Acc.	Prec.	Recall	Acc.	Prec.	Recall	Acc.
KNN-LBP	66	65	72	76	73	77	73	71	76
RF-LBP	68	62	72	80	70	76	73	70	77
DT-LBP	57	58	64	64	60	67	66	65	69
KNN-Gabor	75	71	75	75	74	78	76	79	79
RF-Gabor	66	63	72	81	70	77	79	71	79
DT-Gabor	57	58	65	62	60	66	67	69	72
KNN-HOG	78	72	77	82	80	83	80	74	78
RF-HOG	75	64	72	87	73	79	78	68	78
DT-HOG	55	56	60	59	57	60	60	61	65

Prec. = Precision; Acc. = Accuracy