Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Catch them alive: A malware detection approach through memory forensics, manifold learning and computer vision

Ahmet Selman Bozkir[a,*], *Ersan Tahillioglu*[b], *Murat Aydos*[a], *Ilker Kara*[c]

[a] *Department of Computer Engineering, Hacettepe University, Turkey*
[b] *ASELSAN Inc., Turkey*
[c] *Department of Medical Services and Techniques, Eldivan Medical Services Vocational School Çankırı, Karetekin University, Turkey*

ARTICLE INFO

Article history:
Received 6 June 2020
Revised 16 November 2020
Accepted 28 December 2020
Available online 2 January 2021

Keywords:
Memory forensics
Memory dump
Machine learning
Computer vision
Malware detection
Manifold learning

ABSTRACT

The everlasting increase in usage of information systems and online services have triggered the birth of the new type of malware which are more dangerous and hard to detect. In particular, according to the recent reports, the new type of fileless malware infect the victims' devices without a persistent trace (i.e. file) on hard drives. Moreover, existing static malware detection methods in literature often fail to detect sophisticated malware utilizing various obfuscation and encryption techniques. Our contribution in this study is two-folded. First, we present a novel approach to recognize malware by capturing the memory dump of suspicious processes which can be represented as a RGB image. In contrast to the conventional approaches followed by static and dynamic methods existing in the literature, we aimed to obtain and use memory data to reveal visual patterns that can be classified by employing computer vision and machine learning methods in a multi-class open-set recognition regime. And second, we have applied a state of art manifold learning scheme named UMAP to improve the detection of unknown malware files through binary classification. Throughout the study, we have employed our novel dataset covering 4294 samples in total, including 10 malware families along with the benign executables. Lastly, we obtained their memory dumps and converted them to RGB images by applying 3 different rendering schemes. In order to generate their signatures (i.e. feature vectors), we utilized GIST and HOG (Histogram of Gradients) descriptors as well as their combination. Moreover, the obtained signatures were classified via machine learning algorithms of j48, RBF kernel-based SMO, Random Forest, XGBoost and linear SVM. According to the results of the first phase, we have achieved prediction accuracy up to 96.39% by employing SMO algorithm on the feature vectors combined with GIST+HOG. Besides, the UMAP based manifold learning strategy has improved accuracy of the unknown malware recognition models up to 12.93%, 21.83%, 20.78% on average for Random Forest, linear SVM and XGBoost algorithms respectively. Moreover, on a commercially available standard desktop computer, the suggested approach takes only 3.56 s for analysis on average. The results show that our vision based scheme provides an effective protection mechanism against malicious applications.

* Corresponding author.
E-mail address: selman@cs.hacettepe.edu.tr (A.S. Bozkir).
https://doi.org/10.1016/j.cose.2020.102166

1. Introduction

Computers and other associated computing systems (e.g. mobile devices, IoTs) that we constantly use in our daily lives have become inevitable and rising components of our daily lives, which also attract the attention of cyber-attackers. Thus, computer systems that are used frequently in the sectors such as the defense industry, health, entertainment, banking, and education have been a target for various malicious activities such as illegal profit acquisition, information theft, and denial of service. Gibert et al. (2020) state that the combat between cyber-attackers and security professionals turned out to be an arms race since attackers develop new techniques to evade or subvert anti-malware solutions while the security experts enhance the methods and schemes. According to the AV-Test Institute report, as of 2018, 856 million malware were released (Malware Statistics, the AV-TEST Institute 2019).

To mitigate this problem, several combatting approaches that can be categorized under three branches namely *static, dynamic,* and *memory-based* (Sihwail et al., 2019) were proposed (See Fig. 1). Static methods differ from the others by requiring no runtime analysis based on the execution of suspicious files. Further, this type of analysis utilizes various features such as strings, opcode, API calls, byte sequences, control flow charts that are all revealed from raw bytes of the *portable executables* (PEs) (Gibert et al., 2020). In this context, it can be deduced that *static* methods work with signatures extracted by listed features above. According to Gibert et al. (2020), a signature is an algorithm or a unique hash that differentiates specific malware from the others. As there is no need for execution, static methods require much less computational resources along with providing a fast recognition scheme. However, approaches relying on static analysis often present severe vulnerabilities when code obfuscation and encryption (Or-Meir et al., 2019) come into prominence. Moreover, they can be easily evaded by malware authors when dead code insertion and register reassignment come into play. Thus, their performance involves a high potential for unpredictable decreases in terms of recognition accuracy.

Apart from the static one, the dynamic analysis relies on capturing discriminative patterns from suspicious files sourcing from their behavioral analysis when they are executed in environments such as sandboxes and virtual machines. More precisely, analysis of function calls (Cheng et al., 2017), detection of harmful activities, and manipulation of windows registry entries can be listed as the heading items which the dynamic analysis focuses on. One drawback of this approach is that the dynamic analysis of malware usually requires much more resources (i.e. memory and CPU usage) when compared to static analysis. Moreover, some portable executable (PE) files are required for decompression and unpacking in order to reveal discriminative features. Nonetheless, its performance in terms of accuracy is generally reported higher (Santos et al., 2013). Furthermore, in their comprehensive survey, Or-Meir et al. (2019) have listed some vulnerabilities belonging to dynamic analysis such as sensing the presence of analysis tools to hide its behavior (i.e. by tracing debuggers and signatures created by tools), privilege escalation of malicious executable, nested virtualization and logic bombs. Likewise, they require specific tools for tracing suspicious file activities. Meanwhile, there exist several tools which are designed so as to extract features for dynamic analysis-based methods such as Process Monitor (Process Monitor v3.53, Microsoft 2019), Process Explorer (Process Explorer v16.31, Microsoft 2019), TDIMon (Bryce Cogswell 2019), RegMon, and Wireshark.

Apart from the pros and cons of static and dynamic based schemes, memory-based analysis (i.e. volatile memory forensics) is an efficient way for malware detection that has been gaining more attention in recent years. In principle, volatile memory forensics (VMF) involves two key stages: (a) *acquisition* and (b) *analysis* whereas the acquisition refers to converting data stored in physical or virtual memory into memory dump files via kernel drivers or emulators and the analysis step aims to reveal useful information to detect presence or behavior of malware by smart techniques such as machine learning (Or-Meir et al., 2019). In this perspective, volatile memory forensics presents several key advantages motivating ourselves for this study. As stated in (Dai et al., 2018), data in memory will be inescapably exposed under supervision. Thus, although they can hide via encryption and packing, during the execution all processes become "naked" in the examination point of view since they expose most of the vital information (e.g. registers, code and data segments) to run. In this regard, volatile memory forensics enables the detection of malware by only examining their state in the system RAM. Moreover, another very important advantage of VMF is being robust to fileless malware which avoids detection by leaving no evidence or presence on hard drives (Or-Meir et al., 2019). This new kind of malware resides in the victim's memory until it is terminated or the victim's system shuts down. Due to the lack of footprint, detection of fileless malware is nearly impossible for typical signature-based anti-virus applications.

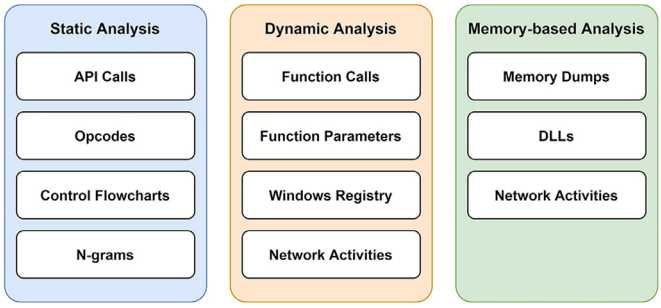

Fig. 1 – Commonly employed malware detection methods in the literature.

Taking all these into account, we propose a vision-based approach employing full memory dumps extracted from processes as the source of information and global image descriptors such as GIST (Oliva and Torralba, 2001) and Histogram of Oriented Gradients (HOG) (Dalal and Triggs, 2005) for the discovery of discriminative visual patterns. Further, we use 5 different machine learning methods to classify whether the suspicious process is benign or malware. In this way, we suggest a more robust scheme against code obfuscations compared to other analysis methods. Moreover, compared to the static analysis we do not need for decompression and unpacking for the analysis.

In the next phase, we have attempted to classify unknown malware samples through binary classification. So we have split the dataset into 3 folds each having variously known malware and benign samples that were used to train classifiers to recognize remaining *unknown* malware files. At this point, we have hypothesized and tested our belief that the code structure and organization of malware files are more similar to other malware types rather than benign files.

To this end, this study mainly presents 5 contributions listed below without any order concern:

- The proposed method in this study focuses on memory analysis that is based on capturing memory dumps which could reflect malware that has been equipped with obfuscation or encryption.
- We have examined the outcome of different byte-to-image rendering schemes in terms of accuracy. To be more precise, we have applied 4 different target image resolutions ranging from 224 to 4096 pixels. Unlike other studies that analyze grayscale images such as (Nataraj et al., 2011, Dai et al., 2018), we have exerted 3-channels RGB images to represent memory dump files.
- Instead of using single image descriptors, we have used GIST and HOG descriptors both solely and together in a manner of information fusion.
- We have applied the state of art manifold learning and dimension reduction technique called UMAP for the first time in the problem domain and evaluated its contribution to unknown malware detection problem.
- We proposed a new dataset containing 10 unique malware families + benign executable class yielding 11 classes. Besides, the dataset we collected involves 3433 training samples as well as 861 validation instances reaching up to 4294 in total.

This paper is organized as follows: In Section 2, we reviewed several related studies. Section 3 introduces the details of the proposed dataset and briefly demonstrates GIST and HOG descriptors. Afterwards, Section 4 details the following sub-procedures: (a) how we have conducted memory dumping operations, (b) how we rendered RGB images, (c) how we obtained discriminative visual features by utilizing GIST and HOG descriptors. Further, Section 5 presents the experimental results and comparison of classification study through 5 unique machine learning algorithms. Finally, Section 6 concludes the study and explains possible future directions.

2. Related work

Currently, there exists a vast amount of studies in every aspect of malware detection and in this section, some of them which focus on static analysis have been briefly reviewed.

In the study proposed by Shijo and Salim, (2015), printable strings (PIS) were extracted from the binary codes and an SVM based learning scheme was used to classify malware files yielding an accuracy of 95.88%. In the work of Santos et al. (2013), they carried out a study comparing both static and dynamic schemes. Hence, opcodes were extracted and computed by term frequency (tf) and frequency of occurrence. In addition, they benefited from features based on dynamic analysis. According to their results, dynamic malware detection poses a higher performance than the static analysis.

In 2019, Bozkir et al. (2019) evaluated several convolutional neural networks on the classification of persistent malware files. For that purpose,they have converted raw binary bytes of portable executables (PEs) into colored images. As the dataset, they utilized Malevis Dataset (The Malevis Dataset 2019) involving 12,394 malware files that have been split up in 8750 as the training set and 3644 as the test set. According to the experiments they performed, they reported the accuracy of%97.48 by using “DenseNet” architecture.

In their study, Yajamanam et al. (2018) investigated and compared the pros and cons of deep learning architectures and GIST descriptors in the domain of malware recognition through image feature extraction. Their findings on rigorous experiments show that these two methods perform equally well in terms of accuracy. Nonetheless, according to (Yajamanam et al., 2018), deep learning methodologies come into prominence by (a) discarding the necessity of extraction of hand-crafted features and (b) a considerable amount of process time.

Nissim et al. (2019) have developed a framework to detect ransomware and RATs on virtual machines that are hosted on cloud systems. They employed MinHash method to analyze similarities among binaries that are created through volatile memory dumps. In their solution, Nissim et al., (2019) have implemented a custom Similarity classifier which is based on MinHash technique and Jaccard coefficient. To improve the detection quality and reduce the time for the process, they have leveraged the method of locality sensitive hashing. The results show that they have achieved a 100% true positive rate along with having a very low false positive rate for ransomware such as 1.8% and 0% for RATs. Their method’s superiority actually sources from analyzing the static information after it was loaded into volatile memory yielding to be able to analyze unpacked, unencrypted clear data.

Shaid and Maarof, (2014) executed malware files within a virtual machine environment, collected user-level API calls and categorized these calls manually according to their suspiciousness levels. Later on, they visualized and categorized those calls as suspicious and non-suspicious ones with hot and cold colors respectively. To this end, they visualized suspicious files based on the behavior they exhibit. Hence, malware behavior can be identified and behavioral images can be used to introduce new ways to malware detection.

Chen, (2018) has carried out the transfer learning technique on the malware detection problem via InceptionV1 deep learning architecture along with grayscaled malware images. Throughout the study, the author performed the experiments over the well known Malimg dataset (Nataraj et al., 2011) having 25 malware classes and Microsoft Malware Classification Dataset (2015) including 9 classes. The results obtained from a multi-class classification regime show that the proposed method together with the softmax classifier has achieved accuracy up to 99.25% with 0.03% false positive rate for "Malimg" dataset. Besides, the obtained accuracy has surpassed the compared classification schemes involving SVM, Random Forest, Naïve Bayes, etc. Chen, (2018) has also tested binary classification on the dataset having 16,518 benign and 10,639 malware files and reported 99.67% accuracy. Nevertheless, we argue their experiments since (1) they have discarded the malware files less than 5 KB in favor of decreasing false positives and (2) the testing strategy has been carried out against already known classes. Instead, the training and testing sets should be mutually exclusive except for the benign family. Additionally, the author pointed out the vulnerability of the method against code obfuscation.

Similarly, Vasan et al. (2020) have taken two deep learning architectures namely VGG16 and Resnet50 and fined tuned them via MalImg dataset to create an ensemble of classifiers. They first applied the transfer learning strategy and reduced the deep features obtained at the end of deep learning models through the dimension reduction technique named principal component analysis (PCA). Vasan et al. (2020) have reported that they reduced the number of dimensions by 90%. According to their results, they have achieved 99% accuracy for unpacked malware whereas 98% for packed ones. However, their study has been conducted by using a dataset having 25 classes in a manner of closed set recognition. Thus, the convolutional neural network models they have trained do not recognize benign samples.

As another study, Yuan et al., (2020) have followed a different way and proposed a solution based on byte-level malware classification through deep convolutional VGG16 model on Markov images. To achieve this feature, they have converted the binary files into Markov images by considering the transfer probability matrices. They have tested their proposal on Microsoft Malware and Drebin datasets (involving 10 classes) and obtained average accuracy rates of 99.26% and 97.36% respectively.

Tam et al. (2015) have taken memory snapshots on an emulator equipped with Android 4.4 version. By using the Volatility Framework, the libraries and strings used by processes extracted to be utilized as a feature. So they have proposed a scalable and portable environment to investigate Android memory. Nevertheless, this study suffers from the size of the small dataset they used.

Dai et al. (2019) have proposed a method that classifies malware files through training artificial neural networks on HOG features extracted from memory dump data. Yet, they have transformed memory dump files into grayscale images recorded in PNG format. As the size of dump data is variable, the sizes of the images have been variable as well. For this reason, the authors have resized them via bi-cubic interpolation to make them have 4096-pixel width if the file size is greater than 4 MB. As a result, they have reached up to an accuracy of 96.7%. Though their study is the closest work in the literature compared to ours, however, our approach involves several differences such as the descriptors and the dataset employed. Furthermore, we have used 3-channel images rather than grayscale counterparts and evaluated different resizing options.

One another study that is closer to a part of our work is the paper of Sharma and Raglin, (2018). In their work, authors of (Sharma and Raglin, 2018) conducted several experiments to measure the efficiency of linear and nonlinear manifold learning algorithms including PCA, Isomap, Diffusion Maps, Laplacian EigenMaps and t-SNE for the task of clustering. The authors have found that nonlinear methods perform better and t-SNE outperforms the other techniques in almost all aspects. The aforementioned study is the first work that has applied manifold learning in the problem domain.

It should be noted that the literature of malware detection also involves very unique approaches such as presented in (Zhang et al., 2014, Zhang et al., 2016). In the former one, the authors have suggested a novel network traffic reasoning approach in order to identify anomalies regarding request-level traffic structures and semantic triggering relations. In this way, they explored a new way to detect even zero-day malware attacks via their new concept so-called "triggering relation discovery". Their experiments based on 6GB+ dataset and conducted with SVM, Bayesian network and Naïve Bayes algorithms have clearly shown that their proposal reaches up to 100% detection accuracy along with serving a scalable solution against DNS bots, spyware and data exfiltration malware. The latter one (Zhang et al., 2016), on the other hand, applies the idea presented in (Zhang et al., 2014) onto Android malware recognition problem by constructing a triggering relation model for dynamic analysis of HTTP traffic generated by Android apps. In other words, in (Zhang et al., 2016), researchers aim to distinguish malicious network requests from benign apps by constructing triggering relation graphs and exploring the root triggers for malicious applications. Their experiments conducted with 14GB+ dataset have shown that the use of triggering relations belonging to network traffic yields 98.2% accuracy.

3. Materials and methods

In this part, we first introduce the dataset we have collected. Next, the employed image descriptors will be demonstrated briefly due to space limitations.

3.1. *Dataset*

It is a well known fact that, for data-driven studies, the importance of well curated and correct dataset are vital. Provided that the literature of pattern recognition based malware detection is reviewed, it can be observed that there exists a limited number of datasets such as "Microsoft Malware Classification Challenge" (Microsoft Malware Classification Dataset 2015) (9 imbalanced classes) and "Malimg" (Nataraj et al., 2011) (25 balanced classes). Bozkir et al. (2019) have recently published another brand new dataset called "Malevis"

Table 1 – The existing malware families and their categories in the dataset.

Class	Category	# of Training	# of Val	Total
Adposhel	Adware	364	93	457
Allaple.A	Worm	349	88	437
Amonetize	Adware	349	87	436
AutoRun-PU	Worm	158	38	196
BrowseFox	Adware	152	38	190
Dinwod!rfn	Trojan	98	29	127
InstallCore.C	Adware	376	91	467
MultiPlug	Adware	390	98	488
VBA	Virus	399	100	499
Vilsel	Trojan	311	78	389
Benign Files	–	487	121	608
	Total	3433	861	4294

(The Malevis Dataset 2019). However, we observed 2 important shortcomings in these datasets. The first one is that they do not contain any available PE files due to security concerns. The latter is that these datasets involve various features or assets belonging to malware classes which causes the lack of benign samples. In this regard, it is obvious that the above-mentioned datasets could be beneficial for close-set inference tasks. To be more specific, without any negative sample (i.e. benign processes) the problem of learning becomes a *close-set problem* whereas we aimed at creating a model that is also capable of determining whether the suspicious process is malicious or not. The existing datasets, therefore, have not been appropriate for our research.

As an attempt to build such a suitable dataset, we cooperated with an information security company located in Turkey. The mentioned company has a special team and system to collect numereous malware samples (i.e. PEs) and they shared the real malware PEs with us. Combined with the received PEs belonging to 10 different malware families we also added an adequate number of benign executables (PEs) mostly taken from the Windows operating system. Note that, the malware classes involved in the dataset were randomly selected. Furthermore, we have added an adequate number of benign samples chosen from digitally signed Microsoft Windows applications. As a result, we built the "Dumpware10" dataset covering 4294 portable executables which are grouped under 11 categories (10 malware + 1 benign class). Apart from having 608 benign instances, the Dumpware10 dataset involves 3686 malware samples collected from different families. The distribution and properties of the dataset have been presented in Table 1. It should be noted that we have partitioned the whole dataset into a training set and validation set, using a split of 80% / 20%. The proposed dataset is publicly available for non-commercial purposes and can be accessed via the link of https://web.cs.hacettepe.edu.tr/~selman/dumpware10/

3.2. *Gist descriptors*

Proposed by Oliva and Torralba, (2001), GIST descriptors were first designed to capture and represent scene pictures in a holistic manner. To be more precise, GIST features provide a low dimensional and discriminative image vector that can be used to identify any image (Oujaoura et al., 2014). In essence, GIST algorithm generates a representation (i.e. spatial envelope) by analyzing the dominant spatial structure of an image by considering 5 different perceptual dimensions (i.e. openness, roughness, naturalness, expansions, and ruggedness,) (Oliva and Torralba, 2001). It should be noted that, apart from scene classification (Oliva and Torralba, 2001), GIST descriptors have been successfully employed in various fields (e.g. recognition of phishing web page (Eroglu et al., 2019) and printed Tifinagh characters (Oujaoura et al., 2014)) as well as malware classification (Nataraj et al., 2011). Thus, in this study, we mainly used GIST descriptors since it (a) is a global image descriptor and (b) can work regardless of the image size.

Through utilizing a pre-determined number of Gabor filters G at different orientations O and scales S, GIST produces a low dimensional feature vector. In order to decrease the computational complexity, the whole image is first scaled and then divided into $B \times B$ blocks yielding vectors containing $B \times B \times G \times O \times S$ dimensions. In fact, for the computation of GIST descriptor, the provided image is first divided into $B \times B$ blocks to avoid loss of information and to reduce computational load (Eroglu et al., 2019). As introduced in (1) and (2), Gabor filters having various scales and orientations are computed for each block.

$$G^{s}_{\theta_i} = Cexp\left(\frac{-\left(x^2_{\theta_i} + y^2_{\theta_i}\right)}{2\sigma^{2(s-1)}}\right) exp(2\pi j(u_0 x_{\theta_i} + v_0 y_{\theta_i}) \tag{1}$$

$$x_{\theta_i} = xcos\theta_i + ysin\theta_i \;\; y_{\theta_i} = -xcos\theta_i + ycos\theta_i \tag{2}$$

By default, for a 3 channel RGB image, the standard GIST descriptor divides the whole surface into 4×4 spatial cells which will be represented two *finer* 8 orientations along with one *coarser* 4 orientations. Consequently, a 960d vector $(3 \times (4 \times 4) \times (8 + 8 + 4))$ is obtained.

3.3. *Histogram of oriented gradients*

Suggested by (Dalal and Triggs, 2005), Histogram of Oriented Gradients (HOG) is a computer vision based method that enables us to capture local object appearance or visual cues by utilizing distribution of intensity gradients and edge directions. Although HOG focuses on local patches, the concatenation of processed image blocks/patches enables researchers to use it as a global image descriptor. Since its proposal, HOG features have been utilized in numerous fields such as pedestrian detection (Dalal and Triggs, 2005) phishing identification (Eroglu et al., 2019; Bozkir and Akcapinar Sezer, 2016) and shape representation. Though it is a well known and powerful method, the shortcoming of HOG descriptor is the obligation of having the same image sizes in order to have a canonical feature vector representation. Nevertheless, we have chosen HOG features like an auxiliary method since it can capture visual cues of the whole image in both local and global perspective

Computation of a HOG vector for an image fundamentally requires the following stages: (1) calculation of gradients, (2) orientation binning and (3) block normalization. Several different types of derivative masks are employed in the phase

of gradient computation. Given a point *(x,y)*, HOG first computes the gradient values in both horizontal and vertical directions by employing $[-1, 0, 1]$ and $[-1, 0, 1]^T$ kernel templates (Dai et al., 2018). As reported in (Dalal and Triggs, 2005), these templates perform the best results when pedestrian classification comes into prominence. Next, as stated in (Dai et al., 2018) the gradients in both *x* and y directions are calculated via the formulas given in (3) and (4) below:

$$grad = \sqrt{G_x(x, y)^2 + G_y(x, y)^2} \tag{3}$$

$$\alpha(x, y) = tan^{-1} \frac{G_x(x, y)}{G_y(x, y)} \tag{4}$$

In essence, to normalize the contrast among the neighborhood regions, the image is divided into a determined number of equally sized *cells* where 8×8 or 16×16 cell groupings constructs a *block*. Note that, these blocks consist of overlapping cells to contribute to the contrast normalization scheme. Throughout the normalization, the obtained local feature vectors belonging to each cell are cascaded based on the voting result. As a result, the parameters such as cell size, block size and image size directly affect the dimension size of the feature vectors. Therefore it is crucial to have a predefined input image size.

3.4. *UMAP - Uniform Manifold approximation and projection for dimension reduction*

In this sub-section, we first briefly outlined what the manifold learning is and concisely introduced the method of UMAP (Uniform Manifold Approximation and Projection). Furthermore, we also presented the reasons behind its selection along with describing its pros and cons compared to other dimension reduction techniques.

Sharma and Raglin (2018) describe the concept of the manifold as a topological space that locally resembles the Euclidean space along with involving a more complex global structure. More precisely, from the local perspective, manifolds can be seen as Euclidean spaces having *homeomorphic* neighborhoods for every point in n-dimensional space. Nonetheless, these points might not be *homeomorphic* from the perspective of the global structure. Spheres, toruses, planes, or folded sheets can be given as examples of manifolds. According to (Sharma and Raglin, 2018), given a dataset having a D-dimensional feature space, revealing the manifold structure lying inside the data is called as manifold learning.

Manifold learning or dimension reduction (DR…) methods deal with discovering new low dimensional embeddings by transforming/mapping the high dimensional data such that the distances among closer data points lie together in the new coordinate system whereas the distant points remain distant by also preserving the parameters. In essence, one significant intuition behind this idea results from two facts: (a) many co-related and overlapping features exist in the datasets and (b) the necessity of avoiding this complexity to achieve a simplified and nonoverlapping representation by preserving the underlying parameters that govern the data (Sharma and Raglin, 2018). One practical implication of these methods is to visualize high dimensional data in 2D or 3D space in order to better discover and investigate the underlying structures, clusters and neighborhoods. From this point of view, manifold learning can be considered as a data transformation tool that employs linear or nonlinear dimensionality reduction techniques.

In this study, we have employed a state-of-art dimension reduction and manifold learning technique named UMAP. UMAP is a relatively new and powerful manifold learning and dimension reduction method that is built on Riemannian and algebraic topology together with fuzzy simplical sets (McInnes et al., 2018). Apart from manifold learning, it also presents many useful features such as clustering, metric learning, visualization and inverse transformation of embeddings. It should be kept in mind that, there exists a vast amount of DR… methods in the literature such as PCA (Jackson, 2005), Laplacian Eigenmaps (Belkin and Niyogi, 2002), Isomap (Tenenbaum et al., 2000), Diffusion Map (Coifman and Lafon, 2006) and t-SNE (Maaten and Hinton, 2008). The t-SNE, among the others, is accepted as the most widely used method (Becht et al., 2019, Ali et al., 2019) and it is often utilized to visualize high dimensional data in 2D or 3D space. Fundamentally, the goal of t-SNE is to discover the patterns by calculating the probability distributions (i.e. Student t-distribution) found in the pairs of high dimensional data points and reflecting them into the lower dimensional space in a way that the distributions are tried to be kept same via Kullback-Leibler divergence (Sharma and Raglin, 2018). Though it is a widely-used and powerful method, it has several shortcomings such as (1) preserving only local neighborhoods (i.e. discarding the global relationships yielding lack of exploring the "big picture"), (2) slow computation that causes scalability problems when large datasets are analyzed, (3) consuming much memory in case of using large perplexity value and (4) lack of producing a learned transformer function to embed new cases when needed. Sharma and Raglin (2018) have applied the above listed DR… methods (except UMAP) in the problem domain of malware detection and found that the best embedding could be achieved by the use of t-SNE. Nonetheless, the recent studies (McInnes et al., 2018, Becht et al., 2019, Ali et al., 2019) comparing UMAP and t-SNE in various fields and datasets report the superiority of UMAP with detailed benchmarks. Although both of these methods follow same or similar methodologies, the main reason behind this finding is that UMAP not only considers the local connectedness but also attempts to preserve data's global structure by utilizing Laplacian Eigenmaps (Kobak and Linderman, 2019)

At its core, the UMAP algorithm initially builds a high dimensional graph structure by utilizing a concept so-called "fuzzy simplical complex" as an analogy to a weighted graph having the weights of edges represent the degree of probability of two data points (i.e. vertex) are related (Coenen and Pearce, 2020). The algorithm decides to connect a data point to another based on whether they overlap within a volumetric range controlled by a diameter parameter. Thus, the selection of the diameter becomes a key component for the optimal trade-off ranging from having very tiny clusters to gluing unnecessary points. At this point, UMAP handles this critical stage by picking a diameter locally by considering the distance of *n*th nearest neighbor of each point in higher dimen-

sional space and the graph gets *fuzzier* along with lowering the likelihood of connections while the *gluing* diameter increases (Coenen and Pearce, 2020). Note that, employment of the *number of neighbors* (NN) is a subtle difference of UMAP compared to the perplexity parameter used by t-SNE. Moreover, UMAP removes the normalization steps for both high and low dimensional probabilities which results in speedup compared to t-SNE. Since UMAP is based on the graph structure, optimization of the layout is an essential step and this is achieved by the Stochastic Gradient Descent algorithm. There exists some advanced mathematical background for the UMAP algorithm and the reader is suggested to review the study (McInnes et al., 2018) for further reading.

In this study, we have incorporated UMAP to (1) improve the binary classification performance at unknown malware recognition problem by obtaining more discriminative lower dimensional embeddings via its supervised metric learning support that enables *warping* the dataset with the help of given class labels; (2) visualize the latent space to investigate and explore the learned lower dimensional embeddings in 2D space. As pointed out by Ali et al. (2019), dimension reduction is a way to improve the efficiency of revealing patterns in datasets. To be more precise about the first argument, UMAP enables to create of a transformer that is trained in a supervised manner through class labels regardless of whether it is a binary or multi-class classification task. In other words, it learns how to embed the samples belonging to different classes from high dimensional feature space into the lower dimensional one by also *skewing* the target feature space. In this regard, we can obtain a well-separated set of new embeddings for each class that could be more isolating and discriminative which yields better and robust accuracy rates. In this perspective, UMAP behaves similarly to variational autoencoders (VAE). However, the experiments of (McInnes et al., 2018) clearly show the superiority of UMAP over VAE. Consequently, we have attempted to investigate the use of UMAP for the first time in the problem domain and explored its contribution to the unknown malware detection problem which was treated in the second phase of our study.

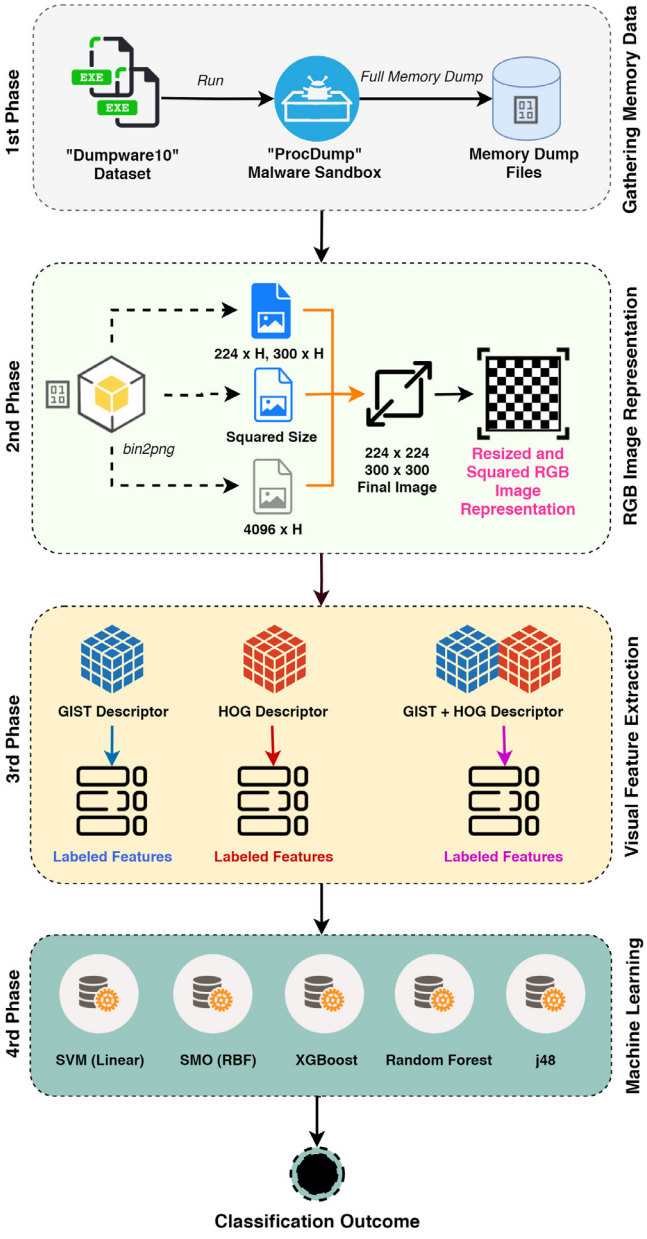

Fig. 2 – The overall workflow of the proposed approach's first phase. Notice that our first phase is composed of four subsequent phases.

4. The approach

In this section, the workflow and system of our whole approach have been explained in detail. The overall study composes of two phases. The followed way in different phases is described in separate sections. The workflow of the first phase is depicted in Fig. 2 in detail. The second phase is illustrated in Fig. 5.

4.1. *Gathering memory data*

If the malware detection studies conducted in recent years are examined, it can be regarded that data located in volatile memory is often used since the memory dumps could store information related to the behavior and structure of portable executables (PE). Moreover, in most of the cases, memory dumps are robust to obfuscation and packing techniques. By definition, the procedure of memory dumping is the extraction of temporary data that exists in the physical and virtual memory of the computer. The memory dump is also referred under different naming conventions such as core dump or system dump, and it is mostly used by software developers during the debug phases. Through memory dumping, data of all processes, or a specific process in physical memory can be extracted. When the target process is dumped, the layouts of thread stacks, text segments, data segments, heap areas, DLL calls of the processes could be revealed. There are quite many tools for the memory dumping process. However, during the study, we have utilized the Procdump (ProcDump v9.0, Microsoft 2019) v9.0 which is a command-line application developed by Microsoft.

Procdump tool enables users to monitor processes and obtain dump files. As being a command-line utility, it was first made available for Microsoft Windows operating system and later on Linux support was provided as well. It is one of the advantages of Procdump that it also enables us to capture virtual address space occupied by processes.

The PE files involved in our Dumpware10 dataset were run in the Windows Sandbox environment shipped with 1903 version of Windows 10 to get their contents located in both physical and virtual address space. In this way, we also protected our physical system against malicious activities. To obtain the dump files, we have executed the Prodcump by providing *–ma* and *–w* startup arguments. The first argument "-ma" tells the tool to gather full memory dump belonging to the target process whereas the "-w" forces the tool to wait for the process until it is run. In our experiments, we first have run Procdump tool and secondly, we have made an intentional delay that has a duration of 5000 ms. As the third stage, the PE file was run and dumping operation finished. The rationale behind the intentional delay is just to ensure that Procdump and the specified malicious application and active. Note that the extension of the generated dump files changes according to the OS of the host. For instance, the Windows version of the Procdump creates .dmp files while the Linux version generates dump files having .vmcore extension. Next, we have located all the .dmp files into their respective folders according to their classes.

One another important point is that the size of dump files varies due to several factors (i.e. dependent DLL files). Besides, the size of dump files generated through the PE files of our corpus ranges between 10 MB to 100 MB. As explained clearly in (Dai et al., 2018), each process consists of several regions in memory space to store different blocks such as DLLs, environmental variables, process heap, thread stack, data segment and text segment as well. Moreover, according to (Korkin and Nesterov, 2015), operating systems employ Address Space Layout Randomized (ASLR) scheme indicating that those blocks are located randomly and fragmented especially when the size of available memory is relatively small. Thus, to collect more consistent and uniform memory dumps we have allocated large enough memory (>16 GB) for the Sandbox environment.

4.2. *Image representation*

The proliferation of binary-to-image conversion invented by Nataraj et al., (2011) has inspired many new anti-malware studies. This kind of visualization of binary files has filled the gap between computer vision and the byte level sequences of executables. Throughout the study, to achieve a suitable representation, we have fundamentally (a) employed RGB based encoding to convert dump files into images and (b) experimented with various column width schemes during these byte-to-image renderings.

The content size of the memory dump data we collected throughout the study is very large and highly variable. Therefore, unlike other studies such as (Nataraj et al., 2011, Dai et al., 2018, Yuan et al., 2020) applying grayscale encoding, we have preferred RGB encoding to generate the images from malware dump files. Note that, in (Bozkir et al., 2019), authors have followed the same way for generating malware images from static malware files. In this regard, each pixel represents 3 sequential bytes yielding a color. The main motivation of this approach is to store more bytes in each row of the image for having better consistency in terms of byte-level alignment. In this way, more information could be placed into each row of the image such that visual similarities belonging to similar samples will be more apparent and easily identifiable. Second, RGB based encoding makes the images more compact since the amount of pixel space will be reduced with a 1/3 ratio yielding less distortion during the post image resizing. Nonetheless, it should be noted that, compared to 8-bit grayscale encoding, this approach could involve disadvantages when byte-level variations come into prominence among memory dumps. Meanwhile, for the byte-to-image conversion, we first updated and modified the Python script called "bin2png" published in https://github.com/ESultanik/bin2png. Next, we have run this script to create the source input images. The mentioned script was written in Python 2.7. However, we modified it that it will be run under Python 3 platform. Meanwhile, instead of lossy JPG format, we preferred the lossless PNG format.

There exist various byte-to-image rendering schemes in the studies of malware detection literature. For instance, in their study, Dai et al. (2018) converted memory dump data into .png encoded gray-scale images having a constant width of 2048 or 4096 pixels according to the file size. However, according to our best knowledge, none of them has conducted a comparative study on the way that these renderings affect classification accuracy. In other words, here we are arguing how one must select the column width for the analysis. As can be seen in Fig. 2, we have selected 4 different column widths such as (1) 224px, (2) 300px, (3) 4096, and finally (4) square root scheme. As the names of the first 3 schemes refer, they correspond to the initial image widths regardless of the memory dump file size. The last scheme (i.e. square root) we applied follows a different strategy. Instead of pre-determined image width, we have computed the square root of the number denoting the (dump file size)/3. Here, we have divided the byte size of PE's by 3 due to the 3-channel color coding and adjusted the edge of the square in a way that it will contain zero-padding elements if necessary. In this way, we obtained 4 different kinds of input images for representing the same training and validation sets. Therefore, as a research question, we also explored the effect of initial image renderings as illustrated in Fig. 3.

The size of initial rendered images allocated quite large even though RGB encoding has been preferred. Further, as can be seen in Fig. 3, the images having 224px width yielded the longest vertical edge. This issue restricts the efficiency of computer vision methods and extends the running time of the visual feature extraction algorithms. Thus, we have transformed them in a way they form a square sized image. Technically speaking, we resized the images throughout their vertical axis via Lanczos interpolation shipped with the OpenCV library (OpenCV Tutorials, OpenCV 2019). Rather than Bi-cubic interpolation used in (Dai et al., 2018), we have employed a better quality interpolation method called Lanczos to reduce the information loss during this unavoidable resize operation. With Lanczos interpolation, the images can be downscaled considering the 8×8 neighborhood in image pixels.

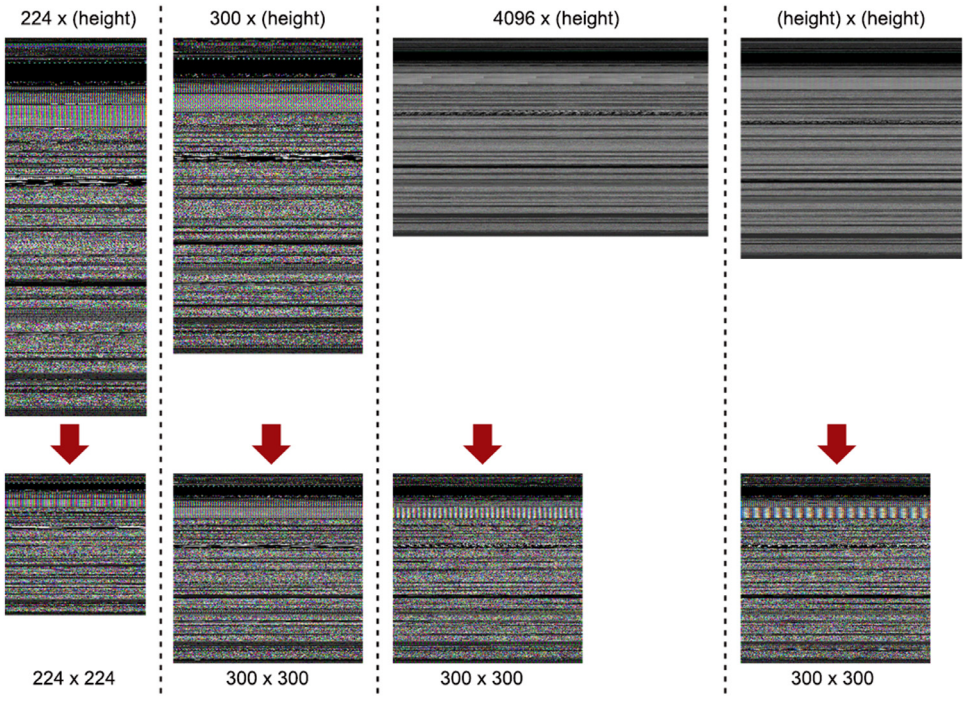

Fig. 3 – Initial image building and resizing schemes we have applied.

The rationale behind the square size transformation is described as follows. As is known, machine learning methods require equal length feature vectors to make classifications, clustering, and so on. Therefore, in substantial number of cases, any computer vision based method must produce equal-sized feature vectors (i.e. signatures) to represent the images. As pointed out in the next sub-section, we extract GIST and HOG based image descriptors during the signature generation stage. Apart from the GIST, due to the nature of the algorithm of Histogram of Oriented Gradients, it needs to process on equal-sized images for producing features having the same length.

Otherwise, some important parameters in the HOG computation such as cell size, number of cells in each block yield varying sized feature vectors. Therefore, it indispensably becomes an obligation to have a canonical image size. It should also be noted that resizing the image into square size undoubtedly causes loss of visual information to a certain extent since this transformation distorts the image. Nevertheless, the previous studies (Dai et al., 2018, Bozkir et al., 2019, Yuan et al., 2020) and our present work show that this loss is not that significant and the discriminative visual cues and patterns could still exist. To this end, we have taken two important advantages: (1) we "equalized" images in terms of size to be used via HOG descriptors and (2) we made the data ready to be modeled with modern convolutional neural networks.

Consequently, we have transformed the byte sequences of the PEs into discriminative visual elements that can be called memory dump images. What is more, is we have investigated the outcome of different pre-processing techniques and compared them.

4.3. *Visual feature extraction*

To extract possible discriminative visual patterns in memory dump images, we have performed two different visual feature descriptors namely GIST and HOG. For getting GIST descriptors we have employed the Python package named "leargist" (Pyleargist 2019). This implementation has been selected due to not only being fast but also enables us to compute colored GIST descriptors having 960-dimensional vectors. Throughout the study, we have used a computer equipped with an Intel 8750 processor and 24 GB memory. The time needed for the GIST feature computation has been measured as 1.67 s on average.

We have also computed the HOG features of the memory dump images by using "Skimage.features" Python package. This package provides a useful set of descriptors such as HOG, SIFT, and Local Binary Patterns. During the HOG feature generation, we have first resized the input images into 256×256. This operation can be considered as downscaling for 300×300 images whereas it can be thought of as an upscaling process for 224×224 images. Following this, we have set the *cell_size* parameter as 32 along with *orientations* as 9 while we have defined the *cell_per_block* argument as 2×2. Besides, we have employed "L2-Normsys" block normalization scheme. As a result, we have obtained $1764d$ feature vectors. The time needed for the HOG feature computation has been measured as 1.12 s on average.

4.4. *Classification via machine learning*

As a result of the feature extraction procedure, vectors representing the characteristic of the images were obtained. To classify the memory dumps according to their visual descrip-

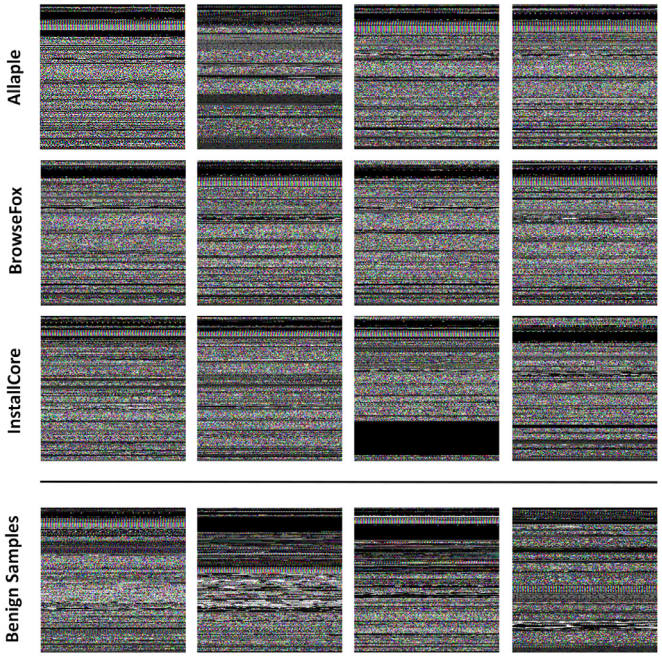

Fig. 4 – Several examples of RGB based memory dump images (224 x 224 renderings) belonging to different malware families and benign samples. The first three rows depict images of various malware classes such as "Allaple", "BrowseFox" and "InstallCore". The last row shows samples belonging to benign PEs. The renderings demonstrate the intra-class and inter-class similarities and variations.

tors, we have employed 5 well know machine learning methods: (1) Random Forest, (2) XGBoost, (3) Linear SVM, (4) Sequential Minimal Optimization, and (5) J48. Three out of these five techniques are in tree-based techniques whereas two of them are involved in structured learning methods. In this way, we aimed to examine whether our problem domain can be learned by tree-based methods and kernel machines. This investigation is important since we observe a high inter-class similarity between some different malware families as can be seen in Fig. 4. On the other hand, Fig. 4 also depicts the intra-class variations belonging to the same malware class.

For an efficient and effective approach, handling this kind of inter-class and intra-class variations/similarities is crucial. Moreover, we believe that this problem must be addressed for any kind of machine learning task. We employed Weka (General documentation, Weka 2019) software to build the models that we evaluated. In particular, we have written a Python script for XGBoost based modeling due to the lack of this method in vanilla Weka distribution. During the experiments carried out with Random Forest (RF), XGBoost, and J48, we have kept the default parameters. For the linear SVM and SMO based modeling, we have set the C value as 10. In particular, we have selected the RBF kernel for SMO based learning process. In this way, we have aimed to observe whether the feature space occupied by HOG and GIST descriptors can be better separated via linear or Gaussian kernel tricks.

It should also be kept in mind that, our problem lies in an open-set classification scheme where the *benign* class also exists. Therefore, having highly discriminant representations and accurate classification of them is vital since most of the samples will come from benign processes in the *wild*.

4.5. *Evaluation metrics*

Throughout the experimental studies, to fairly and quantitatively assess the performance of the built machine learning models, we have operated several metrics such as accuracy rate. The metrics that we have benefited from have been widely used in the literature and provide detailed evaluations of the proposed approaches (Sharma and Sahay, 2014, Eroglu et al., 2019 and Dai et al., 2018). In this sub-section we briefly introduce them as follows:

TP (i.e. true positives) is the measure of artifacts correctly classified as members of positive class whereas FP (i.e. false positives) denotes the number of cases where the model wrongly predicts the positive class. Similarly, TN (i.e. true negatives) shows the number of outcomes where the model determines the ground-truth negative class examples correctly. On the other hand, FN (i.e. false negatives) tells the number of predictions where the created model incorrectly detects the actual negative cases. According to these base definitions we the metrics so-called accuracy, precision, recall, F1-score are defined below in (5), (6), (7), and (8) respectively.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{5}$$

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

Table 2 – GIST based performance comparison for the various classifiers and row length settings. Best configuration was highlighted with bold characters.

ML.Algorithm	Initial Column Width	Feature	Accuracy	FPR	Precision	Recall	F1-Score
Random Forest	224 pixels	GIST	86.06%	0.017	0.867	0.861	0.857
SMO (RBF)	224 pixels	GIST	87.45%	0.014	0.880	0.875	0.875
SVM (Linear)	224 pixels	GIST	85.36%	0.016	0.858	0.854	0.853
XGBoost	224 pixels	GIST	88.26%	0.011	0.885	0.882	0.881
J48	224 pixels	GIST	72.70%	0.029	0.731	0.727	0.726
Random Forest	300 pixels	GIST	89.08%	0.013	0.892	0.892	0.891
SMO (RBF)	300 pixels	GIST	91.40%	0.010	0.915	0.914	0.914
SVM (Linear)	300 pixels	GIST	88.03%	0.014	0.879	0.880	0.879
XGBoost	300 pixels	GIST	88.61%	0.011	0.888	0.886	0.884
J48	300 pixels	GIST	74.09%	0.027	0.756	0.741	0.746
Random Forest	4096 pixels	GIST	93.14%	0.09	0.931	0.931	0.932
SMO (RBF)	4096 pixels	GIST	**94.65%**	**0.006**	**0.948**	**0.947**	**0.947**
SVM (Linear)	4096 pixels	GIST	92.91%	0.08	0.928	0.929	0.928
XGBoost	4096 pixels	GIST	91.63%	0.008	0.919	0.916	0.917
J48	4096 pixels	GIST	79.55%	0.02	0.804	0.797	0.797
Random Forest	Square root scheme	GIST	88.03%	0.016	0.885	0.880	0.878
SMO (RBF)	Square root scheme	GIST	91.75%	0.009	0.917	0.918	0.916
SVM (Linear)	Square root scheme	GIST	88.73%	0.012	0.887	0.887	0.885
XGBoost	Square root scheme	GIST	88.96%	0.011	0.889	0.889	0.888
J48	Square root scheme	GIST	71.19%	0.033	0.724	0.712	0.716

$$F1-score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{8}$$

Moreover, we have also benefited the metric of false positive rate (FPR) which is formalized below in (9) respectively. The FPR measures the proportion of the cases that are wrongly predicted as positive where they are actually negative via dividing the false positives by actual negative cases.

$$FPR = \frac{FP}{Actual\ Negatives} = \frac{FP}{FP + TN} \tag{9}$$

5. Experiments and results

In this sub-section, we have introduced our two-phased experiments and their results along with related discussions.

5.1. *Phase 1 - Identification of known malware families and benign samples*

In this phase, we investigated the performance of GIST and HOG features employed together with different machine learning schemes in the problem domain. These features were previously used in static malware analysis methods (Nataraj et al., 2011, Dai et al., 2018). Nevertheless, we have tested whether their individual and fused usage fits into the problem of dump image classification.

As mentioned before, one of the other concerns of this study is to test and discover the correct image rendering scheme in particular to the problem domain. We, therefore, have prepared 4 versions of the dataset with different image renderings such as (1) 224px, (2) 300px, (3) 4096px and (4) square root scheme. The square root scheme first takes the size of the file and computes the optimal side length of the *square* to fit the content of the file.

Throughout the first phase, we also explored the outcome of the late fusion of the obtained feature vectors. In other words, we concatenated GIST and HOG descriptors to test whether this kind of aggregation yields better classification accuracy. To measure the performance of the machine learning models we have created so far, we have utilized various performance metrics such as accuracy, false positive rate (FPR), precision, recall, and F1-score (the harmonic mean of precision and recall). According to the conducted experiments, the following findings have been found:

(1) Compared to HOG, the GIST descriptor turns out to provided with better results. This shows that the GIST descriptor can generate more discriminative representations than HOG. As listed in Table 2, we achieved an accuracy of 94.65% by using GIST features and the SMO (Rbf) classifier.
(2) When compared to GIST, HOG features have been outperformed in most of the cases. Table 3 shows the classification results related to HOG features and several machine learning methods. Accordingly, we achieved at most 92.68% accuracy and 0.008 false positive rate. The best combination we discovered is HOG + SMO (Rbf). Likewise, the initial image rendering having 4096 pixels width has been identified as the best configuration.
(3) One of the hypotheses that we proposed was the combination of GIST + HOG features yields a better classification scheme since we *lately* fused these two sources of information. According to the results listed in Table 4, this fusion has increased the accuracy and recall scores while reduced the FPR. Therefore, we can conclude that the merging of different visual features yields better classification results regardless of the underlying machine learning method that we have employed.
(4) Among the used ML methods, we explored that SMO with radial basis kernel has outperformed the other methods.

Table 3 – HOG based performance comparison for the various classifiers and row length settings. Best configuration was highlighted with bold characters.

ML.Algorithm	Initial Column Width	Feature	Accuracy	FPR	Precision	Recall	F1-Score
Random Forest	224 pixels	HOG	86.87%	0.017	0.874	0.869	0.866
SMO (RBF)	224 pixels	HOG	90.84%	0.010	0.910	0.909	0.909
SVM (Linear)	224 pixels	HOG	85.71%	0.015	0.859	0.859	0.859
XGBoost	224 pixels	HOG	87.45%	0.012	0.875	0.874	0.872
J48	224 pixels	HOG	67.71%	0.036	0.687	0.677	0.679
Random Forest	300 pixels	HOG	86.06%	0.018	0.861	0.861	0.861
SMO (RBF)	300 pixels	HOG	89.31%	0.012	0.894	0.893	0.892
SVM (Linear)	300 pixels	HOG	85.48%	0.016	0.861	0.855	0.856
XGBoost	300 pixels	HOG	84.90%	0.015	0.848	0.849	0.846
J48	300 pixels	HOG	70.84%	0.031	0.726	0.708	0.712
Random Forest	4096 pixels	HOG	88.61%	0.015	0.892	0.886	0.884
SMO (RBF)	4096 pixels	HOG	**92.68%**	**0.008**	**0.927**	**0.927**	**0.926**
SVM (Linear)	4096 pixels	HOG	88.85%	0.012	0.888	0.889	0.888
XGBoost	4096 pixels	HOG	90.12%	0.010	0.904	0.901	0.900
J48	4096 pixels	HOG	70.84%	0.033	0.717	0.708	0.710
Random Forest	Square root scheme	HOG	88.05%	0.014	0.896	0.889	0.886
SMO (RBF)	Square root scheme	HOG	89.79%	0.012	0.901	0.898	0.898
SVM (Linear)	Square root scheme	HOG	85.13%	0.017	0.857	0.851	0.853
XGBoost	Square root scheme	HOG	88.15%	0.012	0.882	0.881	0.880
J48	Square root scheme	HOG	66.89%	0.037	0.683	0.669	0.674

Table 4 – GIST+HOG based performance comparison for various classifiers and row length settings. Best configuration was highlighted with bold characters.

ML.Algorithm	Initial Column Width	Feature	Accuracy	FPR	Precision	Recall	F1-Score
Random Forest	224 pixels	GIST+HOG	89.89%	0.010	0.899	0.899	0.897
SMO (RBF)	224 pixels	GIST+HOG	94.54%	0.006	0.946	0.945	0.945
SVM (Linear)	224 pixels	GIST+HOG	92.10%	0.009	0.923	0.921	0.921
XGBoost	224 pixels	GIST+HOG	91.86%	0.008	0.922	0.918	0.918
J48	224 pixels	GIST+HOG	73.28%	0.028	0.741	0.733	0.735
Random Forest	300 pixels	GIST+HOG	90.59%	0.012	0.908	0.906	0.904
SMO (RBF)	300 pixels	GIST+HOG	93.14%	0.008	0.932	0.931	0.931
SVM (Linear)	300 pixels	GIST+HOG	90.24%	0.010	0.904	0.902	0.902
XGBoost	300 pixels	GIST+HOG	89.69%	0.010	0.897	0.896	0.895
J48	300 pixels	GIST+HOG	76.42%	0.025	0.781	0.764	0.770
Random Forest	4096 pixels	GIST+HOG	92.91%	0.009	0.933	0.929	0.929
SMO (RBF)	4096 pixels	GIST+HOG	**96.39%**	**0.004**	**0.965**	**0.964**	**0.964**
SVM (Linear)	4096 pixels	GIST+HOG	93.26%	0.007	0.932	0.933	0.932
XGBoost	4096 pixels	GIST+HOG	93.61%	0.006	0.936	0.936	0.935
J48	4096 pixels	GIST+HOG	80.37%	0.022	0.813	0.804	0.806
Random Forest	Square root scheme	GIST+HOG	91.17%	0.011	0.915	0.912	0.911
SMO (RBF)	Square root scheme	GIST+HOG	95.35%	0.005	0.954	0.954	0.953
SVM (Linear)	Square root scheme	GIST+HOG	92.21%	0.008	0.922	0.922	0.922
XGBoost	Square root scheme	GIST+HOG	90.82%	0.009	0.911	0.908	0.909
J48	Square root scheme	GIST+HOG	73.86%	0.028	0.759	0.739	0.746

To avoid overfitting, we have set C (cost) parameter as 10. Thus, we can figure out that the problem in this domain is highly non-linear since the best results have been achieved with non-linear kernels. This finding is also in line with the scores achieved by J48 and RF/XGBoost methods. As J48 is a simpler decision tree method compared to RF/XGBoost, the generalization capacity of it limits the obtained accuracy during inference. Apparently, the use of more than one single tree (i.e. Random Forest, XGBoost) has contributed to separate highly complex feature space (i.e. manifold)

(5) Fig. 5 depicts the confusion matrix of our best classifier which is obtained SMO (with radial basis kernel) classifier along with the use of GIST+HOG features. As can be observed from Fig. 5, the benign class has been found as the most difficult class to predict. Thi has not presented a surprise from our perspective. We believe that this finding is mainly related to the high variance existing in benignware. Also, the structural similarity among the benignware is naturally low. In theory, the open-set recognition problems generally tend to have relatively low accuracy scores regarding the “other” or “unknown” class

Dinwod	AutoRun	Benign	Allaple	BrowseFox	Amonetize	VBA	Vilsel	Multiplug	InstallCore	Adposhel	Predicted / Truth
23	0	5	0	0	1	0	0	0	0	0	Dinwod
0	35	0	0	3	0	0	0	0	0	0	AutoRun
1	3	133	0	0	0	1	0	0	1	2	Benign
0	0	0	88	0	0	0	0	0	0	0	Allaple
0	1	0	0	37	0	0	0	0	0	0	BrowseFox
0	1	0	0	0	85	0	0	1	0	0	Amonetize
0	0	1	0	0	0	99	0	0	0	0	VBA
0	0	1	0	0	0	0	76	0	0	1	Vilsel
0	2	3	0	1	1	0	0	91	0	0	Multiplug
0	0	10	0	0	0	0	0	0	90	0	InstallCore
0	0	0	0	0	0	0	0	0	0	93	Adposhel

Fig. 5 – The confusion matrix obtained from our best classifier (SMO) which utilizes GIST and HOG descriptors (Acc: 0.9639, F1-Score: 0.964). The benign class has found to have the most misclassifications.

(6) We also inferred that the way of building initial image rendering highly affects the classification accuracy. In this study, we have applied 4 different input image size for the problem. In line with the suggestion made by (Dai et al., 2018), 4096 pixels wide rows produce the best results in all experimental studies. Moreover, compared to 224 pixels wide images, the input images initially having 300 pixels outcomes better results in most of the cases.

(7) As an option, we have also investigated the impact of size settings that are highly coupled with the square root of memory dump file size. However, as the results indicate, this approach performs the worst results. Combining with the previous finding, we can conclude that the byte sequences should be tightly aligned to obtain more discriminative visual representations. At this point, we infer that the renderings obtained by the square rooting technique hurt the structural similarity and visual patterns such that the file sizes distort the canonical representation in which a visual descriptor highly benefits from while extracting structural patterns.

(8) The overall execution of the pipeline involving memory dumping, image rendering, feature classification and classification took 3.56 s on average on a standard Intel Core i7 PC equipped with 24 GB of memory and SSD harddrive. Thus, we believe that this duration enables our suggestion to be a viable solution to be used in anti-malware systems.

We have also compared our best results with three other studies such as (Nataraj et al., 2011, Dai et al., 2018 and Rezende et al., 2018). The study of (Nataraj et al., 2011) employed pure use of GIST descriptors along with machine learning methods whereas the (Dai et al., 2018) utilized HOG descriptors to be classified by the multi-layer perceptron model. Likewise, Rezende et al. (2018) suggested the use of deep convolutional neural networks via VGG16 architecture. None of these studies have publicly available source codes to benchmark. So, we have manually coded according to the definitions in the aforementioned papers. To compare with the work of (Rezende et al., 2018), we have utilized the official Pytorch v1.4 library and trained the network for 50 epochs with a decaying learning rate of 0.001 and a batch size of 8. The training process has been done on a computer having an Nvidia Geforce 1050TI graphics card. According to the comparative results provided in Table 5, our work outperforms the first two approaches in terms of accuracy, precision, recall and F1-score. Rezende et al. (2018) which employs deep learning has slightly outperformed our scheme. In this regard, our approach shows a competitive performance compared to (Rezende et al., 2018).

Table 5 – Comparison of our best classifier with other references works.

Study	Accuracy	Precision	Recall	F1
Nataraj et al., 2011	91.40%	0.915	0.914	0.915
Dai et al., 2018	94.54%	0.946	0.945	0.945
Rezende et al., 2018	96.93%	0.970	0.969	0.969
Our best	**96.36%**	**0.964**	**0.964**	**0.964**

According to us, the late fusion of GIST and HOG features enriches the information gained from executable images compared to the single feature employment. Furthermore, it is a well known fact that deep convolutional neural networks (CNN) present a superior performance in almost all aspects of computer vision. Nevertheless, they require a large number of samples in order not to have the overfitting problem. Besides, the nature of the malware detection problem is highly related to capturing distinguishing patterns from the images. In this regard, similar to the findings of (Yajamanam et al., 2018), we argue that conventional CNN architectures do not introduce much accuracy gain since the image variances among malware families do not source from well-known distortions and transformations such as rotation, scaling and other types of affine transformations. On the other hand, it is noteworthy that CNNs enables us to have much faster computation pipelines along with end-to-end learning.

5.2. *Phase 2 - Detection of unknown malware and benign-ware through supervised manifold learning*

As is known, new types of malware families or a variant of known malware are published every day. Therefore, it is becoming more crucial to detect zero-day attacks in time. This fact, in essence, has become the main motivation of this part of the study. In the next phase of our study, therefore, we looked for an answer to the question of "Can we detect a suspicious process as malware even it is not possible to know it's family?". The nature of this question, in essence, leads us to the binary classification.

To find out a robust answer, we have designed a completely different experiment than the first phase. Since our aim is to correctly identify unknown processes, we have reconfigured the train and test sets according to this need. The Dumpware10 dataset which we have created contains 10 malware classes and benign instances. We, thus, decided to split our malware families such that 3 out of 10 families represent the unknown test portion. Accordingly, we targeted to classify the *known unknowns* through the information learned from the remaining malware existing in the training data.

Table 6 – Binary classification performance various ML methods on each fold.

ML.Algorithm	Fold	Training Data				Testing Data			
		Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
Random Forest	Fold 1	100%	1	1	1	62.32%	0.593	0.690	0.550
SVM (Linear)	Fold 1	96.54%	0.961	0.893	0.923	**63.25%**	0.582	0.666	0.549
XGBoost	Fold 1	100%	1	1	1	63.19%	0.598	0.701	0.559
Random Forest	Fold 2	100%	1	1	1	**84.24%**	0.715	0.824	0.746
SVM (Linear)	Fold 2	96.86%	0.955	0.911	0.932	52.29%	0.560	0.622	0.473
XGBoost	Fold 2	100%	1	1	1	59.40%	0.589	0.683	0.531
Random Forest	Fold 3	100%	1	1	1	**82.82%**	0.688	0.769	0.713
SVM (Linear)	Fold 3	96.64%	0.957	0.900	0.926	77.40%	0.652	0.760	0.668
XGBoost	Fold 3	100%	1	1	1	76.24%	0.649	0.764	0.662

To investigate the generalization capability aswell, we have split the dataset into 3 folds in a way that each fold involves 7 malware families for training and 3 malware families for testing. The benign ware set including 608 samples in total have been also split according to the ratio of train/test samples in each fold via random selection. Consequently, the shapes of train/test splits of the folds have become as follows: Fold 1 (Train: 2691, Test: 1603), Fold 2 (Train: 3380, Test: 914) and Fold 3 (Train: 2744, Test: 1549). As the next step, we changed the class labels as "malware" and "benign" in both train and test sets. With this modification, we have converted the multi-class problem into the binary classification scheme. For the classification stage, we have selected linear SVM, XGBoost and Random Forest algorithms having the same parameters as the ones we have used in the previous phase.

The results of the experiments have been given in Table 6. As can be seen from the results, the Random Forest algorithm has been found as the most successful classifier. Moreover, XGBoost and Random Forest algorithms have predicted the training cases 100% correctly. However, linear SVM ($C = 10$) models have failed to identify all the training cases.

For fold 1, the best scheme has been obtained through the linear SVM having achieved a 63.25% accuracy rate along with an F1-score of 0.549. For the fold 2, the Random Forest algorithm has achieved the best performance with the accuracy score of 84.24% and F1-score of 0.746. Similarly, in Fold 3, RF again outperformed the other models gaining 82.82% accuracy and F1-score of 0.713.

Moreover, one another key finding that emerges is that the variance in the metrics between the fold1 and the two others. Fold 1's results are found significantly less than the other folds. As a whole, the results listed in Table 6 are found far less than the scores that we have obtained during phase 1. The actual reason for this outcome is highly related to the imbalance of the classes. Table 7 presents the precision, recall and f1-scores of the predicted benign and malware classes with the most successful algorithms in each fold by also providing the sample counts. Moreover, Fig. 6 also shows the malware splits in each fold with their names.

In light of this new evidences, we can infer (a) benign examples are generally misclassified and (b) a high number of malware samples make the models biased towards making the classifiers behaving in favor of malware class. We can also conclude that the structural similarities between the benign and malware samples make it hard to separate them in high dimensional feature space.

Table 7 – Precision, recall and f1-scores of the predicted benign and malware classes with the most successful algorithms in each fold.

Fold	Class	Precision	Recall	F1	Support
Fold 1	Benign	0.24	0.71	0.35	227
Fold 1	Malware	0.93	0.62	0.74	1376
Fold 2	Benign	0.47	0.80	0.59	130
Fold 2	Malware	0.96	0.85	0.90	784
Fold 3	Benign	0.43	0.69	0.53	220
Fold 3	Malware	0.94	0.85	0.89	1329

At this point, since the problem becomes an imbalanced class distribution problem, there exist several countermeasures that can be taken such as minor class oversampling, major class downsampling or synthetic sample generation for minority class (i.e. benign)

However, these methods have different shortcomings. We, therefore, applied the UMAP manifold learning and dimension reduction method for the first time in the literature to overcome this problem through its supervised metric learning feature. With this feature, we first aimed to enhance the classification performance through obtaining more discriminative and separable lower dimensional embeddings by making the UMAP learn the actual distances among the samples so that it can *warp* the lower-dimensional feature space. Secondly, we visualized the latent space of embeddings to investigate and explore the class distributions on a 2D plane.

To obtain lower-dimensional embedding through UMAP, we have created a Python 3.6 script that imports both UMAP version 0.4.4 and sklearn libraries. To run, UMAP requires several hyper-parameters such as *n_components* (i.e. dimension number of target lower dimensional embeddings), *n_neigbors* (i.e. the parameter to prefer the balance for preserving of local versus global structure and min_*dist* (i.e. the parameter to dictate the algorithm of how tightly it *glues* the data points). The hyper-parameter of min_dist ranging from 0 to 1, actually enables to select the permitted minimum distance among the points in target embeddings. Thus, the larger the value of min_dist broader the preservation of the manifold struc-

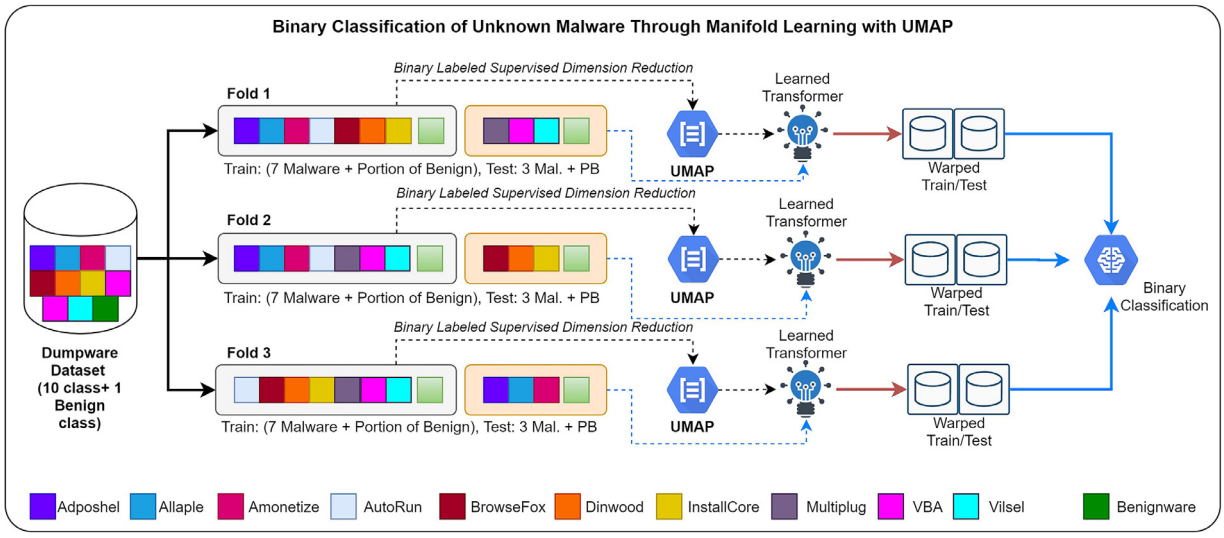

Fig. 6 – The workflow of the "unknown malware detection" phase through the use of UMAP dimension reduction and binary classification. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

ture. One another hyper-parameter is the initial distance metric which Euclidean by default.

As pointed out by Becht et al. (2019), the parameter selection of UMAP is important to have useful embeddings for different purposes. Hence, we have conducted numerous experiments to find out the optimal hyperparameters. In our experiments, we executed a grid-search algorithm that tests the parameter of *n_components* with the values of 4, 5, 10, 25, 50 and 100. Furthermore, we have also sought the optimal *min_dist* parameter with the values ranging from 0.15 to 1 having an interval of 0.05. Apart from these, the *n_neighbors* parameter is tested against values ranging from 5 to 100. Throughout the investigations, we also tested various metrics such as *Euclidean, Manhattan, Chebyshev, Cosine, Jaccard* and *Dice*.

According to the result of these experiments, we found out that our best classification results were obtained by setting the values of n_components = 4, n_neigbors = 55, min_dist = 1 and metric = 'Manhattan'. UMAP applies manifold learning in two different modes (1) supervised, (2) unsupervised. To acquire more separable classes we have preferred supervised mode. In this way, UMAP could learn the topological structure of the classes and embeds them into more separable low dimensional feature space via preserving the intra-class distances controlled by the aforementioned hyperparameters.

Table 8 shows the results after the application of UMAP. The findings we discover from the experiments are listed below as follows:

1. All classifiers become able to correctly classify their training samples
2. Evaluation metrics such as accuracy have risen in different ratios (i.e. Fold 1: 63.25% → 89.95%, Fold 2: 84.24% → 89.27%, Fold 3: 82.82% → 84.95%). More importantly, almost all the classifiers perform equally well on test sets.

We also computed the performance gain in terms of accuracy for each fold and they are ranging from 2.13% to 36.98%.

1. Along with these, we calculated the weighted averages of the accuracy gains for each classification algorithms. In that regard, 4-dimensional embeddings created by UMAP made RF gain 12.93% on average. It is found that, for this particular problem and dataset, the performance of linear SVM and XGBoost have increased by 21.83% and 20.78% respectively.
2. The overall duration to build an embedding transformer object takes 33 s at the test computer equipped with Intel i7 8700 processor.
3. We have depicted the 2D projections of the train/test splits for each fold by presenting their original (raw) feature space and their corresponding new embeddings. Fig. 7 shows the visualizations of each fold in a separate column. The top two projections illustrate the training data (before and after UMAP) whereas the lower two projections of test data present the representations that are before and after UMAP. In the upper part of the figure, the clear separation between the lower dimensional embeddings of two classes can be easily seen for each fold. If we investigate the lower part (i.e. test data) of the figure, we can see the improvement in terms of class separation gets better for all the folds. More precisely, the orange-colored points (i.e. Benign samples) got distant from blue-colored points (i.e. Malware samples) in a way that they build new clusters although some remain as outliers.
4. To better explore the progress which was reflected in classification results in detail, we also presented the new precision, recall and F1-scores for benign and malware classes for each fold. The outcomes are provided in Table 9. According to the results, all the F1-scores showing the harmonic mean of precision-recall values have risen to a certain extend. Beyond this, it seems that UMAP not only improves

Table 8 – Classification performance of various ML methods on each fold after application of UMAP based supervised manifold learning.

ML.Algorithm	Fold	Training Data				Testing Data				
		Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1	Accuracy Gain
Random Forest	Fold 1	100%	1	1	1	89.95%	0.816	0.726	0.760	+27.63%
SVM (Linear)	Fold 1	100%	1	1	1	89.95%	0.816	0.726	0.760	+26.70%
XGBoost	Fold 1	100%	1	1	1	89.95%	0.816	0.726	0.760	+26.76%
Random Forest	Fold 2	100%	1	1	1	89.27%	0.783	0.761	0.771	+5.03%
SVM (Linear)	Fold 2	100%	1	1	1	89.27%	0.783	0.761	0.771	+36.98%
XGBoost	Fold 2	100%	1	1	1	89.38%	0.786	0.761	0.772	+29.97%
Random Forest	Fold 3	100%	1	1	1	84.95%	0.705	0.753	0.724	+2.13%
SVM (Linear)	Fold 3	100%	1	1	1	84.95%	0.705	0.753	0.724	+7.55%
XGBoost	Fold 3	100%	1	1	1	84.95%	0.705	0.753	0.724	+8.71%
Random Forest	Weight. Avg.	–	–	–	–	–	–	–	–	+12,93%
SVM (Linear)	Weight. Avg.	–	–	–	–	–	–	–	–	+21.83%
XGBoost	Weight. Avg.	–	–	–	–	–	–	–	–	+20.78%

Table 9 – Precision, recall and f1-scores of the predicted benign and malware classes for each fold after the application of UMAP.

Fold	Class	Precision	Recall	F1	Support
Fold 1	Benign	0.71	0.48	0.58	227
Fold 1	Malware	0.92	0.97	0.94	1376
Fold 2	Benign	0.64	0.58	0.60	130
Fold 2	Malware	0.93	0.95	0.94	784
Fold 3	Benign	0.48	0.62	0.54	220
Fold 3	Malware	0.93	0.89	0.91	1329

the scores for the minority class (i.e. benign) but also enables the samples of the majority class to reorganize in a way that they are mapped to more concise and clustered fashion.

6. Discussion

Similar to the work of Dai et al. (2018), our proposed scheme mainly relies on the memory dump files. The memory dump content that we deal with in this study is called full memory dumps that involve the data located both in the actual physical memory space and the virtual memory space data on the disk. The employment of memory dump files as the source of information presents two important advantages such as (1) capturing the *exposed* binary content to analyze the process even it is a packed or encrypted file and (2) detection of a new and sophisticated kind of malware so-called fileless malware which injects itself into the system memory without leaving any footprint on disk drives. On the other hand, obtaining memory dumps poses some challenges. First of all, it takes some time in which its duration depends on different environmental parameters and it requires to start on the OS which would risk the system. Therefore, many security software vendors prefer to employ a virtual machine or sandboxes to cope with this risk. Nonetheless, this increases the total time to extract a memory dump and makes the system distant to real-time fashion. In order to overcome this problem some approaches such as the use of Nvidia Cuda enabled GPU hardware for memory dumping has been proposed by (Korkin and Nesterow, 2016). This type of attempts for reducing the time to acquire dump files will lead solutions towards better and close real-time detection. On the other hand, it is known that some smart malware types shutdown themselves immediately when they detect that they are run on a sandbox. This situation brings about other challenges for memory dump based malware detection strategies that need efforts and solutions.

In this study, we have employed a new manifold learning based strategy (i.e. UMAP) to improve unknown malware detection performance and the cross-validation based analysis exhibits that it really improves the accuracy score along with enhancing the precision and recall rates of both the benign and malware classes. The current experiments show that UMAP does not create any bias on the classifers we recreated. However, our dataset named Dumpware10 is limited to 10 malware classes and we believe that our experimental setups require larger datasets to support this argument more strongly.

One another noteworthy aspect is the potentials of deep learning methods that we can benefit from. As we highlighted in the previous sections, the well known and widely used convolutional neural architecture named VGG16 does not contribute the accuracy as we had expected. The authors of the study (Bozkir et al., 2019) conducted experiments on the Malevis dataset (Korkin and Nesterow, 2016) covering 25 malware classes by utilizing several contemporary CNN architectures having different numbers of layers like Alexnet, Resnets, Inception and DenseNets. The results of the study demonstrate that there exists no difference in large margin among the used architectures in the problem domain. As a result, the use of deep CNNs contributes to the problem by providing an end-to-end approach rather than hand-crafted feature extraction. Nonetheless, we believe that state of the art strategies such as attention mechanism equipped CNNs may contribute to the problem domain when the large scale of malware classes come into play. The rationale behind this argument results

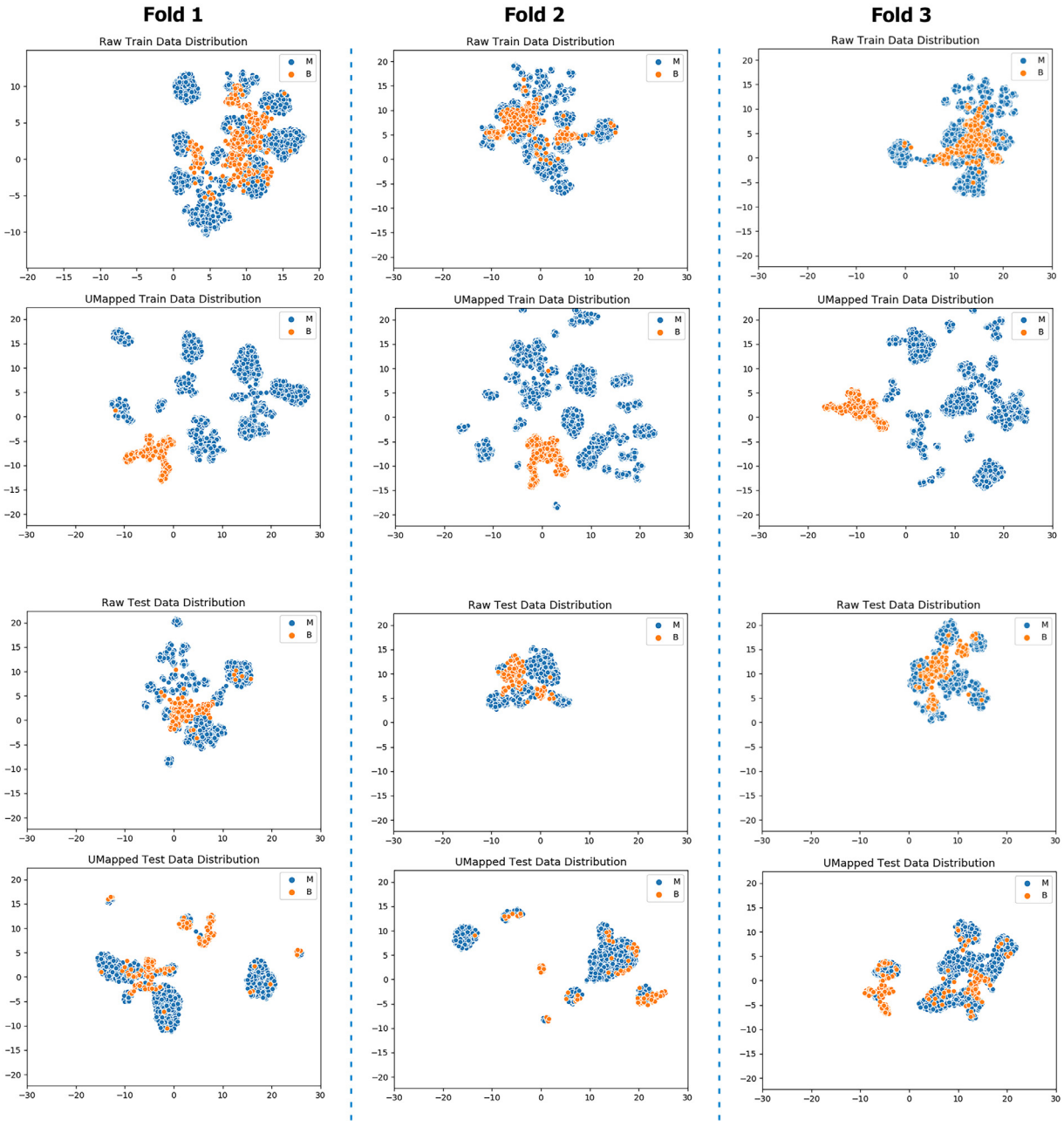

Fig. 7 – The UMAP based 2D visualizations of train/test distributions in each fold that are obtained before and after dimension reduction. (M: Malware, B: Benign-ware). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

from our observations on malware images and the ability of attention mechanisms in modern CNNs focusing on where to look in the images to automatically discover the most discriminative regions. The study (Chen, 2018) explores the distinguishing parts of the malware images via superpixels – a segmentation method for images – and shows that not every pixel contributes to the classification performance. Therefore, for better separation among the classes, the future studies need to identify the essential and discriminative regions by discarding the unnecessary visual information that resembles the curse of dimensionality.

7. Conclusion

In this study, we have conducted a two-phase study to identify and detect malware and benign executables by utilizing two image descriptors namely GIST and HOG to analyze memory dump images. Moreover, in the second phase, we attempted to employ the state of the art manifold learning technique named UMAP for the first time in the field to improve the robustness of classifiers which is meant to better decide whether a suspicious process in a computer's memory is malicious or not. The experiments conducted on the embeddings

obtained with UMAP shows that it is an appropriate method that can be used in the domain to contribute both to the problem of class imbalance and feature visualization. In this sense, we suggested an approach following the volatile memory forensics to build a robust scheme against fileless malware. Thus, memory dumps of processes have been utilized as the main source of information. So as to support the study findings, we have prepared and published a publicly available dataset having instances for 10 malware families also including benign samples. Moreover, we have explored the impact of various image rendering schemes and found that 4096px column width yields the best results. The results indicated that when transforming memory dumps to initial images, having larger columns (i.e. larger width values) result in having better representations. Furthermore, the late fusion of GIST and HOG features yields the best outcome.

Consequently, the proposed approach having 96.39% accuracy shows promising results through also offering a reasonable time to detect such as 3.56 s. We believe that the volatile memory forensics will gain more popularity in the near future due to the increase in fileless malware and similar threats.

As future work, to increase the accuracy and to obtain an end-to-end analysis platform, we plan on investigating the abilities of convolutional neural networks that are equipped with self-attention mechanisms to reveal more discriminative representations as well as achieving better generalization capabilities. One another interesting idea that we project to test is the inverse transformation feature of UMAP which might enable us to generate synthetic image features analogous to the decoding of auto-encoders.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Ahmet Selman Bozkir: Conceptualization, Methodology, Software, Formal analysis, Writing - original draft. **Ersan Tahillioglu:** Data curation, Software. **Murat Aydos:** Project administration, Validation, Supervision, Writing - review & editing. **Ilker Kara:** Validation, Writing - review & editing.

REFERENCES

Ali M, Jones MW, Xie X, Williams M. TimeCluster: dimension reduction appliedto temporal data for visual analytics. Vis. Comput. 2019;35.

Becht E, McInnes L, Healy J, Dutertre C-A, Kwork I-W-H, Ng LG, Ginhoux F, Newell EW. Dimensionality reduction for visualizing single-cell data using UMAP. Nat. Biotechnol. 2019;37.

Belkin M, Niyogi P. In: Advances in neural information processing systems. Laplacian eigenmaps and spectral techniques for embedding and clustering; 2002.

Bozkir AS, Akcapinar Sezer E. In: 4th International Symposium on Digital Forensic and Security (ISDFS). Use of HOG Descriptors in Phishing Detection Arkansas, USA; 2016.

Bozkir AS, Cankaya AO, Aydos M. In: Signal Processing and Applications Congress, SIU, Sivas. Utilization and Comparison of Convolutional Neural Networks in Malware Recognition; 2019.

Chen L, "Deep Transfer Learning for Static Malware Classification", arXiv:1812.07606, 2018.

Cheng Y, Fan W, Huang W, An J. A Shellcode Detection Method Based on Full Native API Sequence and Support Vector Machine, vol. 242. IOP Publishing; 2017.

Coenen A, Pearce A, "Understanding UMAP", https://pair-code.github.io/understanding-umap/, Technical Note, (Available online at 27.5.2020), 2019.

Coifman RR, Lafon S. Diffusion maps. Appl. Comput. Harmon. Anal. 2006;21(1).

Dai Y, Li H, Qian Y, Lu X. A malware classification method based on memory dump grayscale image. Digital Investigation 2018;27:30–7.

Dai Y, Li H, Qian Y, Yang R, Zheng M. SMASH: a Malware Detection Method Based on Multi-Feature Ensemble Learning. IEEE Access 2019;7:112588–97.

Dalal N, Triggs B. Histograms of oriented gradients for human detection, vol. 1; 2005. p. 886–93.

Eroglu E, Bozkir AS, Aydos M. Brand Recognition of Phishing Web Pages via Global Image Descriptors. Eur. J. Sci. Technol. 2019 Special Issue.

General documentation, Weka. (2019). [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/.

Gibert D, Mateu C, Planes J. The rise of machine learning for detection and classification of malware: research developments, trends and challenges. J. Network Comp. Appl. 2020;153.

Jackson JE. A User's Guide to Principal Components, 587. John Wiley & Sons; 2005.

Kobak D, Linderman GC. UMAP does not preserve global structure any better than t-SNE when using the same initialization; 2019. biorXiv preprint bioRxiv2019.12.19.877522.

Korkin I, Nesterov I. In: TheADFSL Conference on Digital Forensics. Applying memory forensics to rootkit detection. Security and Law; 2015.

Korkin I, Nesterow I. In: 11th ADFSL Conference on Digital Forensics, Security and Law. Acceleration of Statistical Detection of Zero-day Malware in the Memory Dump Using Cuda-Enabled GPU Hardware; 2016.

Maaten Lvd, Hinton G. Visualizing data using t-sne,. J. Mach. Lear. Res. 2008;9.

Malware Statistics, the AV-TEST Institute. (2019). [Online]. Available: https://www.av-test.org/en/statistics/malware/.

L. McInnes, J. Healy, J. Melville, "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction", arXiv preprint arXiv1802.03426, 2018.

Microsoft Malware Classification Dataset, (2015). [Online]. Available: https://www.kaggle.com/c/malware-classification.

Nataraj L, Karthikeyan S, Jacob G, Manjunath BS. Malware images: visualization and automatic classification. Proceedings of the 8th international symposium on visualization for cyber security. ACM, 2011.

Nissim N, Lahav O, Cohen A, Elovici Y, Rokach L. Volatile memory analysis using the MinHash method for efficient and secured detection of malware in private cloud. Computers & Security 2019;87.

Oliva A, Torralba A. Modeling the shape of the scene: a holistic representation of the spatial envelope. Int. J. Comput. Vis. 2001;42:145–75.

OpenCV Tutorials, OpenCV. (2019). [Online]. Available: https://opencv.org/.

Or-Meir O, Nissim N, Elovici Y, Rokach L. Dynamic Malware Analysis in the Modern Era – A State of the Art Survey. ACM Comput. Surv. 2019;52(5).
Oujaoura M, Minaoui M, Fakir M, El Ayachi R, Bencharef O. Recognition of Isolated Printed Tifinagh Characters. Int. J. Comput. Appl. 2014;85.
ProcDump v9.0, Microsoft. (2019). [Online]. Available: https://docs.microsoft.com/en-us/sysinternals/downloads/procdump.
Process Explorer v16.31, Microsoft. (2019). [Online]. Available: https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer.
Process Monitor v3.53, Microsoft. (2019). [Online]. Available: https://docs.microsoft.com/en-us/sysinternals/downloads/procmon.
Pyleargist, (2019). [Online] Available: https://pypi.org/project/pyleargist/.
Rezende E, Ruppert G, Carvalho T, Theophilo A, Ramos F, de Geus P. In: Advances in Intelligent Systems and Computing. Malicious Software Classification Using VGG16 Deep Neural Network's Bottleneck Features; 2018.
Santos I, Devesa J, Brezo F, Nieves J, Bringas PG. In: International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions. Opem: A static-dynamic approach for machine-learning-based malware detection. Berlin, Heidelberg: Springer; 2013.
Shaid SZM, Maarof MA. In: 2014 International Symposium on Biometrics and Security Technologies (ISBAST). Malware behavior image for malware variant identification. IEEE; 2014.
Sharma PK, Raglin A. In: 17th IEEE International Conference on Machine Learning and Applications. Efficacy of Nonlinear Manifold Learning in Malware Image Pattern Analysis; 2018.
Sharma A, Sahay SK. Evolution and detection of polymorphic and metamorphic malwares: a survey. Int. J. Comput. Appl. 2014;90:7–11.
Shijo PV, Salim A. Integrated static and dynamic analysis for malware detection. Procedia Comput. Sci. 2015;46:804–11.
Sihwail R, Omar K, Zainol Ariffin KA, Al Afghani S. Malware detection approach based on artifacts in memory image and dynamic analysis. Appl. Sci. 2019;9.18:3680.
Tam K, Edwards N, Cavallaro L. In: Engineering Secure Software and Systems (ESSoS) Doctoral Symposium. Detecting Android malware using memory image forensics; 2015.
Tenenbaum JB, De Silva V, Langford JC. A global geometric framework for nonlinear dimensionality reduction. Science 2000;290(5500).
TDIMon, Mark Russinovich. &; Bryce Cogswell. (2019). [Online]. Available: https://sysinternals.d4rk4.ru/Utilities/TdiMon.html.
The Malevis Dataset, (2019). [Online]. Available: https://web.cs.hacettepe.edu.tr/~selman/malevis/.
Vasan D, Alazab M, Wassan S, Safaei B, Zheng Q. Image-Based Malware Classification using Ensemble of CNN Architectures (IMCEC). Computers & Security 2020;92.
Yajamanam S, Selvin VRS, Di Troia F, Stamp M. Deep learning versus gist descriptors for image-based malware classification. Icissp 2018:553–61.
Yuan B, Wang J, Liu D, Gao W, Wu P, Bao X. Byte-level Malware Classification Based on Markov Images and Deep Learning. Computers & Security 2020;92.
Zhang H, Yao D(Daphne), Ramakrishnan N. Detection of stealty malware activities with traffic causality and scalable triggering relation discovery. Proc. of the 9th ACM symposium on information, computer and communications security (ASIA CCS'14), 2014.
Zhang H, Yao DD, Ramakrishnan N. Causality-based Sensemaking of Network Traffic for Android Application Security. Proc. of the 2016 ACM Workshop on Artificial Intelligence and Security (AISec'2016), 2016.

AHMET S. BOZKIR was born in Muğla Turkey, in 1983. He received a B.S. degree in computer engineering from Eskişehir Osmangazi University in 2002. Besides, he received M.Sc. and Ph.D. degrees in computer engineering from the Hacettepe University in 2009 and 2016 respectively. He is currently working as a Research Assistant at Hacettepe University Multimedia Information Laboratory (HUMIR). He has more than 33 studies covering fields such as Information Security, Human-Computer Interaction, Information Retrieval, Machine Learning and Engineering Geology.

ERSAN TAHILLIOGLU works as an embedded software engineer in ASELSAN Corporation. He has received his B.S. degree in computer engineering from Hacettepe University in 2016. He is currently an M.Sc student in the same department. His-research interests include topics such as computer security and machine learning. His-current research specifically focuses on malware detection utilization of memory forensic and machine learning.

MURAT AYDOS. Dr. Murat Aydos received the B.Sc. degree from Yildiz Technical University (Turkey) in 1991, and M.S. degree from Electrical and Computer Engineering Department, Oklahoma State University (USA), in 1996. He completed his Ph.D. study at Oregon State University, Electrical Engineering and Computer Science Department in June 2001. Dr. Aydos joined Informatics Institute at Hacettepe University in April 2013. He is the Head of Information Security Division at the Informatics Institute. Dr. Aydos is the author/co-author of more than 30 technical publications focusing on the applications of Cryptographic Primitives, Information & Data Security Mechanisms.

ILKER KARA has been an Assist. Prof. in the Department of Medical Services and Techniques, Eldivan Medical Services Vocational School in Çankırı Karatekin University since 2019. He has also been a part-time lecturer in the Computer Engineering Department of Hacettepe University since 2017. Kara completed his Ph.D. at Gazi University as of 2015. His-research interests cover digital investigation, malware analysis and internet security. He has actively collaborated with researchers from several other disciplines such as computer science and forensics security in particular. He is currently working as the head of Information Security Division at the Informatics Institute. Besides, Dr. Kara authored more than 20 technical publications focusing on the applications of Cyber Security, Malware Analysis; Data Security Mechanisms.