Computer Networks 201 (2021) 108595

Contents lists available at ScienceDirect

Computer Networks

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

Distinguishing malicious programs based on visualization and hybrid learning algorithms

Sanjeev Kumar [*], B. Janet

Department of Computer Application, National Institute of Technology, Tiruchirappalli, India

ARTICLE INFO

Keywords:
Malware classification
Malware attacks
Machine classifier
Convolutional neural network
Data processing
Honeypot

ABSTRACT

Modern malware threats demand a robust and scalable detection system. This paper presents a novel proactive monitoring and analysis architecture called malware threat intelligence system (MTIS) to collect and classify real-world samples of modern malware families. Microsoft Window‘s portable executable (PE) files are systematically labeled using clustering and AVClass engine. These labeled malware samples are visualized into grayscale images, which are further utilized for the extraction of textural features. This paper uses local descriptors (LBP, DSIFT, GLCM) and global descriptors(GIST) to extract local and global textural features from a grayscale image. Malware images are distinguished using a hybrid approach of machine learning methods and a proposed convolutional neural network (CNN) employed with early stopping configurations. Results have demonstrated that the proposed architecture detects modern and real-world malware samples with better classification accuracy without code reversing and domain expertise. Four machine learning algorithms, namely, K-nearest neighbors (k-NN), Support vector machine(SVM), Naive Bayes(NB), and Random forest, are compared systematically, with different image resolutions and data split, enabling efficient model selection. Deep learning of MTIS (DL-MTIS) is compared with former methods and with CNN+GIST, CNN with the whole image as inputs, and k-NN+GIST method, and it is observed to be superior in performance. The results also reveal that the proposed method is resilient to packed and encrypted malware samples.

1. Introduction

Over the years, usage of the internet has grown exponentially. The number of connected devices (IoT, Mobiles, SCADA systems) to the internet has increased many-fold. The adoption of internet-enabled services such as e-banking, e-commerce, social media, etc., makes end-users digitally connected. Even organizations have started providing work from home (WFH) connected through the internet during the pandemic. As per a report [1], nearly 5.3 billion people will be using the internet by 2023. With this rapid growth of the internet, the cyber-attack risk is a primary concern, making malware detection a complex problem. Moreover, attackers have started to adopt new tactics to damage computer security. As per Symantec‘s threat report [2], malware authors used coronavirus emails embedded with malicious software to infect end-users systems. Such types of modern and complex malware threats demand a new robust and scalable malware detection system. Fig. 1 shows the categorization of malware into different malware types such as ransomware, worms, trojan viruses, rootkits, etc. Intruders design malicious programs to harm computers and internet-connected device‘s security and privacy. Malware detection and classification into respective families have been an exciting research area in recent years because modern malware‘s authors are equipped with new technologies to create complex malware.

Static and dynamic analysis are two commonly used malware detection techniques. The static analysis relies on signatures, i.e., rule sets, to characterize the malware binary. On the other side, dynamic analysis executes binary samples in a stack of virtualized machines or in a sandbox to record behavioral traces of malware. It extracts program’s behavior artefacts such as registry changes, file changes, API call monitoring, network connections [3]; these behavior artefacts are further used to characterize malware features. Traditional machine learning-based malware detection methods used feature vectors extracted either through static or dynamic analysis. Static features can be byte code sequences, static opcode patterns, ASCII strings, DLL function calls, etc. [3]. Dynamic features of malware can be API calls (list of API calls, arguments, categories, etc.), system calls, network traffic, registry, process changes, etc., which can be extracted through run-time execution of malicious programs.

In recent years, the use of visualization-based malware analysis has overcome the time and computational overhead of static code reversing and dynamic analysis. Representing malware samples as a visual image

[*] Corresponding author.
E-mail address: 405918054@nitt.edu (S. Kumar).

https://doi.org/10.1016/j.comnet.2021.108595
Received 11 July 2021; Received in revised form 8 October 2021; Accepted 31 October 2021
Available online 9 November 2021

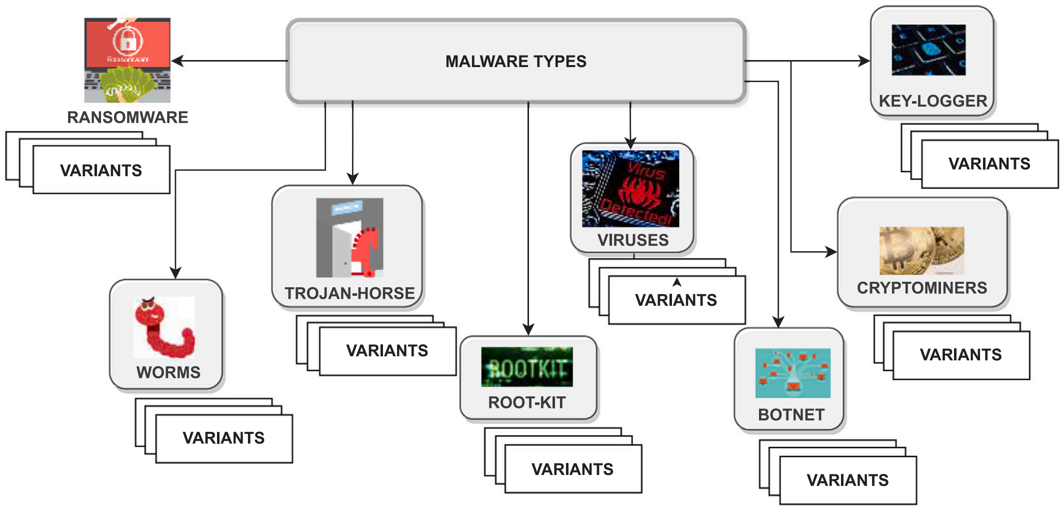


Fig. 1. Malware types.

enables researchers and the industry to leverage image processing techniques for malware detection. The image content of malware typically consists of local and global features. Local features describe intriguing details of an image that distinguish it from other images like blobs, points, edge pixels, or tiny image patches. Whereas global features [4] give the gist of the whole image like contour details, texture, shape description, etc. The main benefits of using static visual analysis are that it is less dependent on security feature engineering and requires fewer resources than static code reversing and dynamic analysis. With image visualization, images belonging to the same malware family appear very similar in layout and textural patterns.

Challenges: The following are broad challenges in the current malware detection technologies:

1. The current state-of-art malware detection technologies heavily depend on anti-virus software that requires signature databases to detect malicious patterns. However, such software is unable to detect new or unknown malware. Also, the AV scanners are unable to detect packed and encrypted malware. Even mere reusability of the code with some packers can produce a new variant of malware that can bypass signature-based techniques.
2. Malware detection using traditional machine learning algorithms uses features extracted through dynamic or static analysis. There is no doubt that machine learning significantly accelerates malware detection‘s performance, but these methods rely solely on manually crafted features that are not scalable. In addition, extraction of these features is very domain-dependent and time-consuming, making it unsuitable for detecting modern complex malware.
3. Visualization-based approach for malware detection can handle polymorphic code obfuscation such as packing/encryption and environment detection methods used by malware variants. These approaches reduce dependence on domain experts and cut manual feature engineering, eventually saving time. But still, these methods need improvements to detect real-world and modern malware samples, which are usually very imbalanced. Merely using single textural features, [5–7] of a malware image will not be sufficient to detect real-world modern malware samples that are highly cohesive.

None of the earlier research on malware detection has tested modern and real-world malware in a honeypot context to handle continuous collection and classification of the latest malware. Most existing works are tested on public benchmarked datasets which are outdated, very limited in volume and variety. During research and development of a malware collection system [8,9], collected PE files have to be classified instead of relying on anti-virus-based detection or traditional machine learning methods. Modern malware samples are highly cohesive, and utilizing a single image descriptor does not reveal the complete patterns of a malware family. To determine the similarities between different families would be difficult by using a limited type of textural features. Moreover, the performance of a learning algorithm is determined by labeled training data. Previous literature has not addressed the flexibility in extracting image features (local and global) for different image ratios, selecting training datasets, and classifying models. Also, existing research works in this area have not addressed the overfitting issue that most machine learning models face.

This research paper brings the following contributions to address issues as mentioned above: (a) A novel architecture as malware threat intelligence system (MTIS) is proposed to collect and classify malware attacks in Microsoft Window‘s honeypot environment, (b) The proposed system pre-processes and labels the collected binary executables through systematic methodology instead of relying only on AV technology, (c) A visual analysis based approach is presented to detect malware using hybrid machine learning and deep learning algorithms, (d) A systematic study using both local and global textural features of visualized malware image with respect to training and testing ratios is presented, (e) Intensive experiments are performed to build best classification model and validated on real-world, modern malware samples collected on a proactive malware collection system. (f) The proposed method neither relies on static analysis nor dynamic analysis-based feature engineering; hence it does not require domain expert knowledge: like reversing binary files or assembly language. The proposed malware classification approaches are resilient to packing and encryption.

The research paper is organized as follows: Section 2 presents related work in malware detection. Section 3 discusses a few background topics. Section 4 details the proposed malware threat intelligence system (MTIS) architecture along with its sub-modules. Section 5 presents materials and datasets used in this research followed by experiment results in Section 6. Section 7 concludes the research work with future scope.

2. Related work

2.1. Static code analysis and dynamic behavior features based malware detection

Several works of literature have been conducted that utilized malware features through static code analysis and dynamic behavior analysis. Table 1 gives a summary of research conducted towards malware

Table 1
Summary of static and dynamic analysis for malware detection.

Work	Classifiers	Features	Data count	Data split	Acc(%)
Static analysis literatures					
[10]	RIPPER,NB, Multi-Classifier	DLL, DLL Functions, GNU Strings, Byte Sequences	3,265 malicious, 1,001 clean	5-fold cross validation	97.11
[11]	SVM, LR,RF, ANN, DT, BDT, NB,BNB.	Static OpCode	5,677 malicious, 20,416 benign	K-fold cross validation	95
[12]	DT,SVM,k-NN, BN	Opcode sequences	13,189 malware, 13,000 benign	90:10	92.92
[13]	RF	PE Optional Header fields (PEFs),	338 malware, 214 benign	80:20	97%
[14]	SVM	Static taint analysis	2,866 benign, 15,338 malicious apps	10-fold cross validation	
[15]	LSTM	Opcode sequence patterns	969 malware, 123 benign	70:30	97.56
[16]	SVM	Trigram opcode instructions	399 malwares	70:30	89.47
Dynamic analysis literatures					
[17]	Increment clustering	API calls	3,133 reports of 24 families		
[18]	SVM	API call sequences	4,288 samples of 9 families	40:10:50	
[19]	SVM	Android IPC System calls High level behavior OS objects	5246 samples		84
[20]	RF,SVM NB, DT, k-NN	permission, app resources, system calls	34,343 samples 17,365 benign apps, 16,978 malware		97
[21]	Online machine learning	API calls	17,400 malware, 532 benign		98
[22]	Rule based (YARA)	List of APIs,, API sequences, Network traffic	604 Windows Portable Executable (PE32)	70:30	97.22
[23]	SVM, NB, DT, k-NN and RB	API calls Memory dumps	1200 PE malware, 400 benign.	10-fold cross validation	98.5
[24]	LR,RF,DT,NB	API calls API arguments	90 benign apps, 90 malicious apps	80:20	98.4
[25]	LSTM	API sequences	7107 malicious software	80:20	95
[26]	Attention model	API sequences	9192 normal, 27770 malware samples	10-fold cross validation	97.23
[27]	SVM, RF,XGB, MLP,CNN	Printable (ASCII) strings	500,000 samples	10-fold cross validation	99.2
[28]	DNN	manifest properties , API calls	18,000 benign and malicious samples	80:10:10	96.78

detection using static or dynamic analysis features. In terms of static analysis, Schultz M. et al. [10] introduced data mining techniques using static elements (DLL, DLL‘s function call, Strings, Byte sequences) of malware. Santos, Igor, et al. [12], Elkhawas et al. [16], Shabtai, A. et al. [11] and Lu et al. [15] used static opcode patterns of malware as feature map to distinguish them. Belaoued et al. [13] presented real-time malware detection based on CHI-Square Test and PE-File features. The advantage of using static analysis is that these methods are quite accurate and fast as they do not require a resource-intensive execution environment, but they suffer from evasion tactics such as obfuscation, packing/encryption, code transformation, etc.

In terms of dynamic analysis, several authors used dynamic features such as API call sequences, system calls, etc. Rieck, Konrad, et al. [17], Lin, Chih-Ta, et al. [18], Dash, S. K. et al. [19], Pektaş et al. [21], Dai et al. [29], Belaoued, Mohamed, et al. [22], Sihwail, Rami, et al. [23], and Ali, Muhammad, et al. [24], Chen, Jun, et al. [26] used API calls as a feature map to build various machine learning classifiers. Dai et al. [29] also utilized low-level features of malware such as hardware features, API call sequences from memory dumps.

Feng, Ruitao, et al. [28] presented an android malware detection system known as MobiTive, which leverages customized deep neural networks. They used manifest properties and API calls directly from binary code instead of decompiling APK files. However, they used a limited feature map (manifest properties, API calls) to be extended to make a more robust detection. Mimura et al. [27] proposed NLP-based malware detection in a practical environment using printable (ASCII) strings from malicious and benign samples. Cai, Haipeng, et al. [20] used the diverse set of dynamic features such as permission, app resources, or system calls. Kim, TaeGuen, et al. [30] proposed a multimodal deep learning method using multiple feature types of android malware. The authors used multiple features such as strings, opcode, API, shared library functions, permission features, components, and environmental features. They used several deep neural networks based on the types of the feature map.

Dynamic analysis is potentially more resilient to evasion strategies, i.e., code obfuscation, packing/encryption, but they are highly environment-dependent and resource-intensive. When malware detects its environment, it can deviate its behavior. Additionally, behavior-based methods usually utilize a stack of virtual machines; malware authors can employ VM detection, environment detection methods to avoid the complete execution of malware. Furthermore, dynamic analysis of modern malicious programs may be ignored because the execution environment does not comply for them to execute. Overall, both static and dynamic analysis methods are highly computationally expensive and domain-dependent.

2.2. *Image visualization for malware detection*

Several studies have been proposed using visualization-based malware detection. Nataraj et al. [31] introduced image-visualization-based malware detection using grayscale images. They represented 8-bit vectors in grayscale images; several works have been proposed, motivated by this way of using features of a visualized image. Narayanan, B. et al. [32], Naeem et al. [5], Naeem, Hamad, et al. [6], Liu, Ya-shu, et al. [7] have extracted textural features of a visualized malware image before applying classification algorithms. However, feature extraction of these methods is very limited; either they have used GIST for global feature extraction or exploited DSIFT for local feature extraction.

Cui et al. [33] developed CNN-based malware detection combined with BAT algorithm as a data equalization method to avoid overfitting. However, input image size is fixed in their proposed work, thus making the technique unscalable for varying image sizes. Ni et al. [34] presented malware identification using SimHash and CNN model. They used LSH (locality-sensitive hashing) to convert similar hash values later into grayscale images. They utilized a static approach that requires further study to detect packed and encrypted malware effectively, also mentioned in their future work.

Kalash, Mahmoud, et al. [34] proposed deep convolutional neural network-based malware classification. They obtained excellent classification accuracy at 98.52% for the Malimg dataset and 99.97% for Microsoft dataset [34]. However, they do not discuss the model‘s overfitting problem, which exists in most neural networks. Sun et al. [45] also presented deep learning-based malware detection using image features. They combined static analysis with CNN and RNN models. However, their proposed method is not validated on large-scale and real-world malware datasets.

Rezende, Edmar, et al. [36], Dai, Yusheng, et al. [29],Davuluru et al. [37], Zhao, Yuntao, et al. [38], Naeem, Hamad, et al. [39], Cui, Zhihua, et al. [40], Ding, Yuxin, et al. [42], Huang, Xiang, et al. [43], Catak, Ferhat Ozgur, et al. [44] used CNN models for automatic feature extraction and classification to detect malware samples that were visualized into images. Similarly, Vasan, Danish, et al. [41] proposed a fine-tuned deep CNN model for malware classification that leverages colored malware images. However, fine-tuning a deep convolutional neural network is a computationally expensive approach. Table 2 lists the research works that leveraged malware image visualization for their detection.

Table 3 shows a comparative analysis of existing literature concerning their advantages and disadvantages. Most visual-based malware analysis methods utilized limited textural features of a visualized image. None of the existing research work gives the analyst or anti-virus industry an option to select the learning model depending on the type of textural features vs. the learning approach. The existing works of visual-based malware analysis do not have an integrated and continuous data collection methodology, which requires a comprehensive approach for malware detection [36,43]. Moreover, very few research works have discussed the overfitting problem in neural networks while working with smaller and unevenly distributed datasets.

3. Background

3.1. *Convolution Neural Networks*

A convolution neural network (CNN) is simply a feed-forward neural network model comprised of a set of neurons with learnable weights and biases. This model has recently gained much attention for its ability to extract feature maps from an input image without human intervention and has been used widely in image processing. The neural network has many nodes called neurons which are interconnected. Every node of a neural network has an activation function to produce the output.

The mathematical equation of convolutional layer is expressed as:

$$F_{r,c} = F\left(\sum_{m}\sum_{n} W_{m,n} X_{r+m,c+n} + W_b\right) \tag{1}$$

Where $X_{r,c}$ represents row r, column c of the input image’s pixel, and $W_{m,n}$ depicts row m, column n of the filter’s pixel. W_b depicts the bias in a convolutional neural network’s layer, and F represents the activation function of a convolutional layer. ReLU activation function (Eq. (2)) is used in the hidden layers, and softmax activation function is used in output prediction layer. Function $F_{r,c}$ depicts the feature map of row r, column c.

$$F(z) = max(0, z) \tag{2}$$

The softmax activation function (Eq. (3)) is a generalized logistic function that takes an input range of score $x \epsilon R^n$ as a vector and predicts the probability of output vector $p \epsilon R^n$ at end of the architecture.

$$Prob = \begin{pmatrix} P_i \\ . \\ . \\ P_n \end{pmatrix}, Prob_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{3}$$

Table 2
Summary of image-visualization based malware detection.

Work	Visualization	Image ratio	Datasets	Data split	Acc(%)
[31]	Grayscale	64×64	MalImg	90:10	98
[32]	Grayscale	256×16	Microsoft	90:10, 80:1:1	96.6
[33]	Grayscale	24×24,48×48, 96×96,192×192	MalImg	–	94.5
[34]	Grayscale	Different size	Microsoft	80:20	98.86
[35]	Grayscale	–	Malimg, Microsoft	90:10	98.52, 99.97
[36]	Grayscale	224×224	VirusSign	90:10	92.97
[29]	Grayscale	–	27k malwares of 214 families	10:20	98.1
[5]	Color image	200×200 192×192	Leopard datasets, Malimg	ML: Different ratios, DL:70:30	ML:95.11%, DL-98.186
[6]	Grayscale	Different size	Malimg, Malheur, VirusShare, Microsoft	Different ratios	98.4
[7]	Grayscale	–	MalingA, MalingB, CNCERT Labs.	10-fold cross validation	99.0
[37]	Grayscale	64×64	Microsoft	72:8:20	98.4
[38]	Grayscale	–	Microsoft		92
[39]	Color image	224×224 229×229	Leopard Mobile dataset, Malimg	70:20	98.79
[40]	Grayscale	50×50, 100×100 120×120	Malware image data from Vision Research Lab.	–	97.6
[41]	Color image	224×224	Malimg, IoT-android mobile dataset	70:30	98.82
[42]	Grayscale	512×512	DREBIN	–	95.1
[43]	Grayscale	512×512	Virussign.com	80:20	94.7
[44]	Color image	–	5762 samples (308 benign)	80:20	98
The Proposed Method	Grayscale	48×48,64×64, 128×128, 224×224	Malimg, Real-world malware samples	Different ratios	ML:98.28 DL:98.71

This paper uses the cross-entropy loss function (Eq. (4)), which is commonly used in deep neural networks.

$$L(x, y) = -\left[y \log(x) + (1-y)\log(1-x)\right] \tag{4}$$

The Root Mean Square Propagation (RMSprop) (Eq. (5)) in an adaptive learning rate algorithm, used in CNN architecture learns iteratively by error propagation.

$$SdW = \beta s_{dW} + (1-\beta)\left(\frac{\partial J}{\partial W}\right)^2, W = W - \alpha \frac{\frac{\partial J}{\partial W}}{\int s_{dW}^{corrected} + \epsilon} \tag{5}$$

where S is average of past gradients, $\frac{\partial J}{\partial W}$ is the cost gradient for current layer weight W, β is tuned hyperparameter, α defines learning rate, and ϵ very small value to avoid dividing by zero.

3.2. *t-SNE visualization*

T-distributed stochastic neighbor embedding(t-SNE) is a very popular technique used to scale high dimensional data into lower-dimensional feature space [46]. This algorithm works by calculating Euclidean distances for each data point in high-dimensional space based on similarities and maps it into its conditional probabilities such that it can be projected in low-dimensional space [46]. This study uses t-SNE to quantifies data separability numerically. Clear separation in t-SNE space indicates that classes are distinguishable from one another. The gist features of grayscale images of size 192×192 for both the datasets used in this research are visualized using a t-SNE plot. Fig. 2 shows feature visualization with each node as one malware sample and each malware family depicted by a different color. From Figure Fig. 2, it is observed that similar malware families grouped into a single color, hence indicating correct data labeling.

3.3. *Evaluation method*

This study evaluates the implemented models using standard metrics, including accuracy, precision, recall, F1, and ROC curves. Given that,

True Positive (TP): is the number of samples correctly detected as malware

False Positive (FP): is the number of samples incorrectly detected as malware

True Negative (TN): is the number of samples correctly detected as benign

Table 3
Comparison of existing literatures with the proposed work.

Work	Visual features (Local/Global/ Hybrid)	Data labeling (Y/No)	Address Model overfitting (Y/N)	Classifier model (ML/DL/ Hybrid)	Integrated Data Collection (Y/N)
[31]	Global (GIST)	✗	✗	ML (KNN)	✗
[32]	Global (PCA)	✗	✓Partial (Data balancing)	Hybrid (ANN KNN SVM)	✗
[33]	CNN	✗	✓Partial	DL(CNN)	
[34]	CNN	✗	✓Partial	DL(CNN)	✗
[35]	CNN	✓Partial (AVClass)	✗	DL (CNN)	✗
[36]	CNN	✓Partial (Virustotal)	✗	ML (SVM)	✓ (VirusSign)
[29]	CNN	✗	✗	ML (Voting)	✗
[6]	Hybrid (DSIFT,GIST)	✗	✗	ML (KNN, SVM, NB)	✗
[7]	Local (LBP,DSIFT)	✗	✗	ML (KNN,RF)	✗
[37]	CNN features	✗	✗	ML (KNN,SVM)	✗
[38]	CNN features	✗	✗	DL (CNN)	✗
[39]	CNN features	✗	✗	DL (CNN)	✗
[40]	CNN features	✗	✗	DL (CNN)	✗
[41]	CNN features	✗	✓ (Data augmentation)	DL (CNN)	✗
[42]	CNN features	✗	✗	DL (CNN)	✗
[43]	CNN features	✗	✗	DL (CNN)	✓ (VirusSign)
[44]	CNN features		✓ (Data augmentation)		✓
The Proposed method	Hybrid	✓ (AVClass, Clustering)	✓ (Data augmentation, early stopping)	Hybrid (SVM, KNN, NB,RF, CNN)	✓ (Honeypots)

Note: ML-Machine learning, DL-Deep learning.

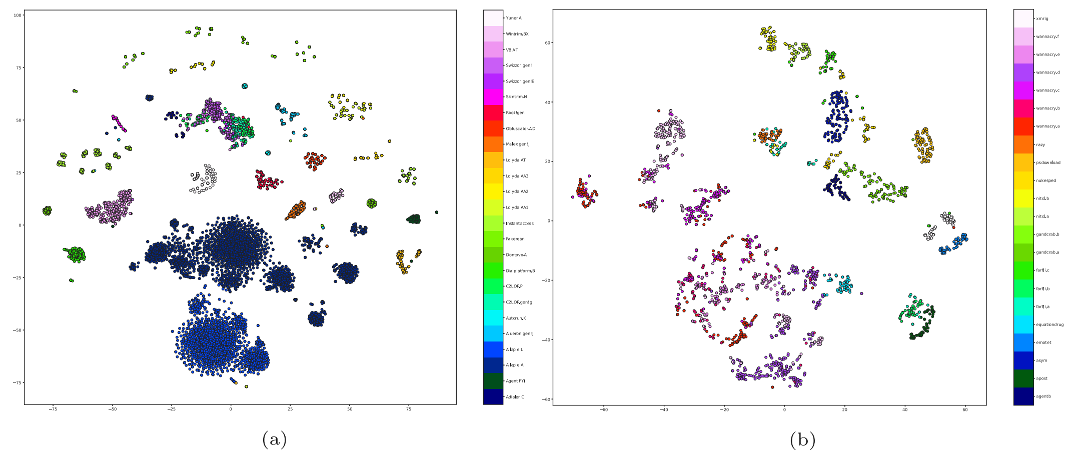


Fig. 2. t-SNE visualization plots.

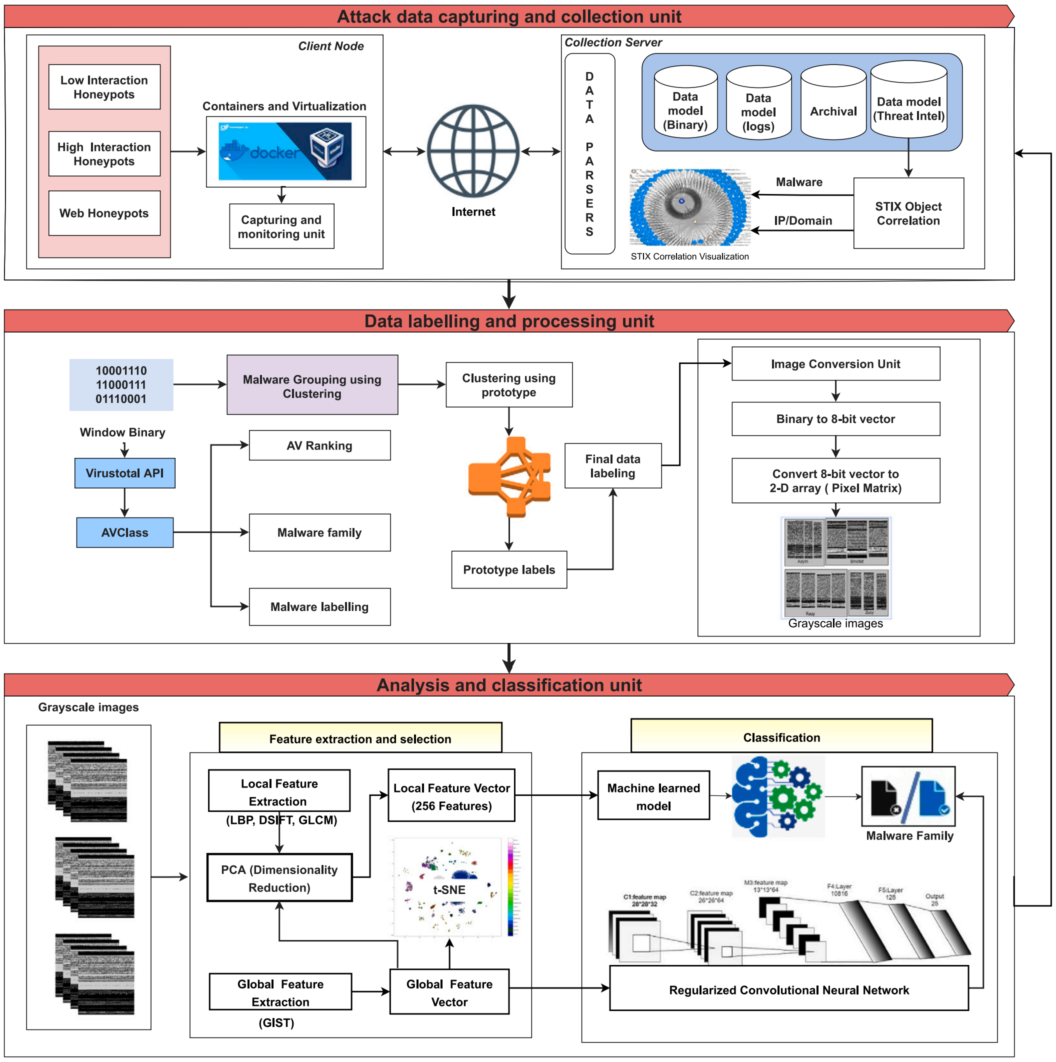

Attack data capturing and collection unit
Client Node
Low Interaction Honeypots
High Interaction Honeypots
Web Honeypots
Containers and Virtualization
Capturing and monitoring unit
Internet
Collection Server
DATA PARSERS
Data model (Binary)
Data model (logs)
Archival
Data model (Threat Intel)
Malware
IP/Domain
STIX Object Correlation
STIX Correlation Visualization
Data labelling and processing unit
10001110 11000111 01110001
Window Binary
Virustotal API
AVClass
Malware Grouping using Clustering
Clustering using prototype
AV Ranking
Malware family
Malware labelling
Final data labeling
Prototype labels
Image Conversion Unit
Binary to 8-bit vector
Convert 8-bit vector to 2-D array (Pixel Matrix)
Grayscale images
Analysis and classification unit
Grayscale images
Feature extraction and selection
Local Feature Extraction (LBP, DSIFT, GLCM)
Local Feature Vector (256 Features)
PCA (Dimensionality Reduction)
t-SNE
Global Feature Extraction (GIST)
Global Feature Vector
Classification
Machine learned model
Malware Family
Regularized Convolutional Neural Network

Fig. 3. The proposed architecture of malware threat intelligence system.

False Negative (FN): is the number of samples incorrectly detected as benign

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN},\quad Precision = \frac{TP}{TP+FP} \tag{6}$$

$$Recall = \frac{TP}{TP+FN},\quad F1 = 2 * \frac{Recall * Precision}{Recall + Precision} \tag{7}$$

The receiver operating characteristic (ROC) plot shows two parameters: true positive and false-positive rates, also known as TPR and FPR, which can be expressed as:

$$TPR = \frac{TP}{TP+FN},\quad FPR = \frac{FP}{FP+TN} \tag{8}$$

4. Architectural model for malware threat intelligence system (MTIS)

Fig. 3 shows the design architecture of the proposed malware threat intelligence system (MTIS). Broadly, it has three units, i.e., attack data capturing and collection unit, data labeling, and processing unit, analysis and classification unit. For attack data capturing and collection, this research extends earlier work of [8,9]- a multi-platform honeypot supporting different operating systems, protocols, and vulnerable configurations for attack data capturing and collection. It is a client–server architecture where the client node consists of various honeypots using para-virtualization and full virtualization technologies. The client node consists of a capturing and monitoring unit to log the network traffic(PCAP), Window‘s Portable Executable(PE) files. The client code compresses the monitored data (Network PCAP and PE files) and transfers it to a central collection server for further storage and analysis. The central server consists of a custom parser for processing and ingesting the collected data into corresponding binary databases, network logs, system logs, and threat intel data models.

4.1. Data labeling and processing unit

The data labeling and processing unit perform systematic and accurate labeling of malware families before applying classification algorithms because the performance of classifiers depends upon the labeled datasets. Collected PE files are processed through AVclass [47] and customized Malheur engine [17]. AVClass performs malware labeling by applying a majority rule on reported malware labels from multiple anti-virus vendors, as perceived on VirusTotal report [47].

Further, this unit utilizes a customized Malheur engine for prototype-based clustering of malware samples. The PE files collected on designed and operationalized MTIS architecture are fetched and submitted to cuckoo sandbox API in an automated way for dynamic

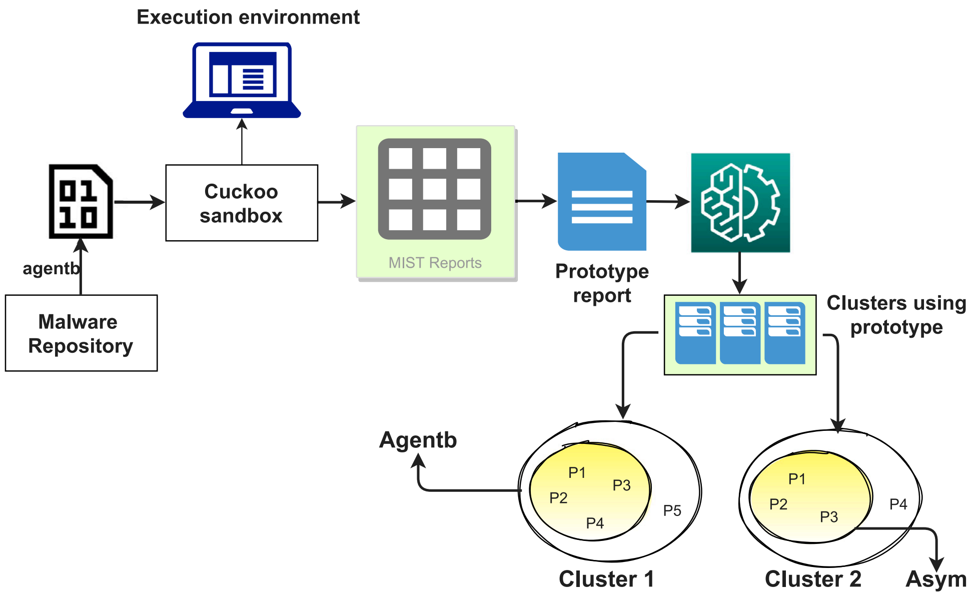


Fig. 4. Data labeling procedure using clustering.

behavior analysis for 180 s. The processing module embeds the sample‘s malware instruction set once the run-time execution is completed in the cuckoo sandbox (MIST). The monitoring and execution environment for PE files is cuckoo sandbox version 1.3, VirtualBox version 5.1, Window‘s 7 32-bit architecture, and ten virtual machines (VM) instances.

Malheur engine selects prototypes iteratively from a set of MIST reports which represents whole datasets. Then, it performs clustering using these extracted prototypes. Once the clusters using prototypes are created, the hash of each prototype of a cluster is labeled with the AVClass engine. This study considers those malware counts commonly marked by AVclass and Malheur engine to get the consistently labeled dataset. Algorithm 1 shows the basic procedure of data labeling. The majority of samples of a particular cluster belonging to the same malware family are considered; Hence, only those samples are counted, which are labeled by AVclass and lie in a single cluster. If a cluster has some samples of the different malware family, they are not taken into final labeling and discard them as an outlier of a cluster. For example, malware class agentb has five clusters as C000–C001, C000–0022,C000–C0023, C000–0045, and a rejected cluster with corresponding prototype as 1,949,3,11,14. The prototype of 949 has a label as Siscos, while other clusters' prototypes have been labeled as agentb. Therefore, malware data in clusters 1,3,11,14 are considered agentb, and samples as 949 prototypes are not counted. The real dataset represents Window‘s executables of 22 malware families. Fig. 4 shows building blocks of the data labeling procedure.

Further, the labeling unit has a data conversion module that converts labeled PE files into a grayscale image. This paper uses binary visualization of executable files [31]. The contents of a given binary executable file are first represented in an 8-bit binary code comprised of 0‘s and 1‘s. Then with the help of standard binary to decimal conversion, the 8-bit code is represented as a 1-D array of unsigned decimal integer values ranging from 0 to 255. For example, let 0011010101111110…, be the binary string which can be considered as 8-bit binary substrings i.e 00110101, 01111110, …and the equivalent decimal representation be $(00110101)_2 = (53)_{10}$ and $(01111110)_2 = (126)_{10}$. For an image conversion procedure, a 1-D array is reshaped into a 2-D pixel matrix, further providing the grayscale image.

```
   Input: PE files collected on MTIS $D_i|i = 1, 2....m$
1  for each binary PE sample $B_i|i = 1, 2....n$ do
2      Execute each binary in Cuckoo sandbox;
3      Get MIST reports $M_i|i = 1, 2....m$ of each binary;
4      Extract prototypes $P_i|i = 1, 2....j$ of each MIST reports
       ($j > m$);
5      Perform clustering using extracted prototypes;
6  end
7  if clusters $C_i|i = 1, 2....m$ of malware binary B1 consists of
   prototypes $P_i|i = 1, 2....j$ then
8      Get hash of each prototype of a cluster;
9      Get label D1 of the prototype hash through AVClass;
10     Take all samples of the corresponding cluster into label D1;
11     Reject different labels and do not consider them for the
       training dataset;
12 else
13     Execute model with more samples.
14 end
```

Algorithm 1: Procedure for data labeling.

4.2. Analysis and classification unit

The analysis and classification unit has feature extraction and selection, machine learning, and deep learning classifiers. As mentioned in Section 2, a hybrid textural feature map of a visualized malware image is used in this research. The learning algorithms consist of a hybrid ML and DL-based classification method corresponding to the type of feature map. The workflow of each step is highlighted in subsequent sections:

4.3. Local feature extraction and selection

The local descriptors such as local binary patterns(LBP), a gray level co-occurrence matrix (GLCM), dense scale-invariant feature transform (DSIFT) describe local features of an image. After converting a PE file into a grayscale image of size 192 × 192, feature maps of an image are extracted through local descriptors (Table 4). Feature map of LBP and GLCM is quite lower in size; hence they do not require further reduction. In comparison with other descriptors, DSIFT has higher dimensions and is more accurate. With this advantage, DSIFT

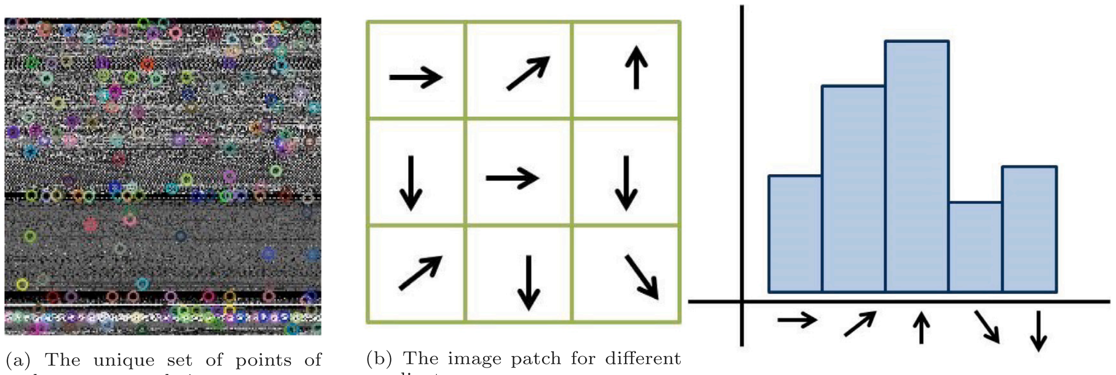

(a) The unique set of points of malware gray scale image.
(b) The image patch for different gradients.
(c) Gradient into Histograms.

Fig. 5. DSIFT procedure.

and GIST used in this research have obtained higher accuracy over LBP and GLCM.

DSIFT descriptors describe dense local features of the malware image. These features are invariant to image transformation like rotation and scaling. It outputs a unique set of key points with their descriptors as whole image is reduced to a set of uniquely identifiable points. The descriptor gives details about locally derived set of points. Image patches surrounding the unique points are taken for a 4×4 neighborhood location grid that is 16 neighborhood pixels. These surrounding patches around the key points describe any change in the intensity values of the image or the gradients in the local neighborhood region. This gives us the descriptor vector about the key points. The distribution of gradients (that gives the viewpoint changes) in the local region around the key point is converted into the image histogram computed in 45° orientations (into eight bins). Then we obtain a total of 128 dimensions ($4 \times 4 \times 8$) out of the image feature. Fig. 5 shows the complete workflow of the DSIFT descriptor. The total dimension of the DSIFT feature is given in Eq. (9):

$$D = ceil\left(\frac{W-(4-1)*\eta_x}{\delta_x}\right) * ceil\left(\frac{W-(4-1)*\eta_y}{\delta_y}\right) * 128 \qquad (9)$$

Where W is the width of the malware image; δ_x is the horizontal step and δ_y is the vertical step; η_x and η_y are the height and width of the cell. In the current implementation, the image width is 192, the step size (Delta) value is 13 and the width of the neighborhood image patch cell is 1. The total DSIFT feature dimension turns up to 225×128. The features are flattened, and a total of 28800 features are extracted. Further, the dimension of the extracted features is reduced to 256 feature maps using principal component analysis [32].

•*The high dimensional features, DSIFT and GIST are further reduced into lower dimensions using PCA. PCA technique transforms the N-dimension vector's feature map into a vector of M-dimensions with $N > M$. In this paper, the dimensions of DSIFT, and GIST descriptor's feature map are reduced into 256 feature vectors [32].*

4.4. Global feature extraction and selection

To extract global textural features, conventional algorithms are GIST descriptor, a wavelet transformation, and a Gabor transformation [6]. Global features define the structural information about a malware image. In this research, the most common global descriptor known as GIST is used [4,31,32].

4.5. Classification-Machine learning methods for malware classification

For machine learning-based classifications, the four most popular algorithms, namely, K-nearest neighbors (k-NN), Support vector machine(SVM), Naive Bayes(NB), and Random forest, are used in this study [6,7,32]. Experiments are performed towards the selection of the best classifier. The extracted local and global features of malware images are fed to machine learning models with different training ratios (Table 7).

Table 4
Feature map of different image descriptors.

Windows malware dataset (MalImg)		
Descriptor name	Feature space	Image shape
LBP	10	192×192
DSIFT	28800	192×192
GLCM	36	192×192
GIST	960	192×192
DSIFT+PCA	256	192×192
GIST+PCA	256	192×192

4.6. Deep learning method for malware classification

This paper proposes a convolution neural network (CNN) employed with early stopping configurations for malware classification using extracted global features. It consists of two convolution layers, a max-pooling layer, a dropout layer, a fully connected layer, and a softmax layer. In this implemented CNN model, the first convolution layer has 32 kernels, and the second layer has 64 kernels; both have kernel sizes of 3×3. Max pooling layer with pooling size of 2×2 and dropout rate of 0.25 are taken in the proposed architecture. After flattening the outputs from previous layers, it is passed to the fully connected layer of 128 neurons with a ReLU activation function. Finally, the output layer with the softmax function predicts the final target class. The structure of the proposed CNN architecture is shown in Fig. 6.

An early stopping technique based on the validation loss during the model's process is employed. It helps the model generalize well on test instances; in other words, it prevents the overfitting of the model. As per literature, all conventional neural networks are generally prone to overfitting [48]. Early stopping of the model handles this problem by monitoring the validation loss in each epoch of the model's training process. The definition of the early stopping technique employed in the model is as follows:

Definition 4.1 (*Early Stopping*). $E_{val\_loss}(Condition)$ as the minimum validation error until the condition is met that indicates to stop the training of the model. The condition is defined that for each epoch, the validation error at that iteration is matched with the next ith number of epochs, i.e., patience=q. Early stopping occurs when minimum validation loss is found among the next ith number of epochs. A best "Condition" of a particular run is one with a minimum validation error, and error does not improve till the next ith epochs.

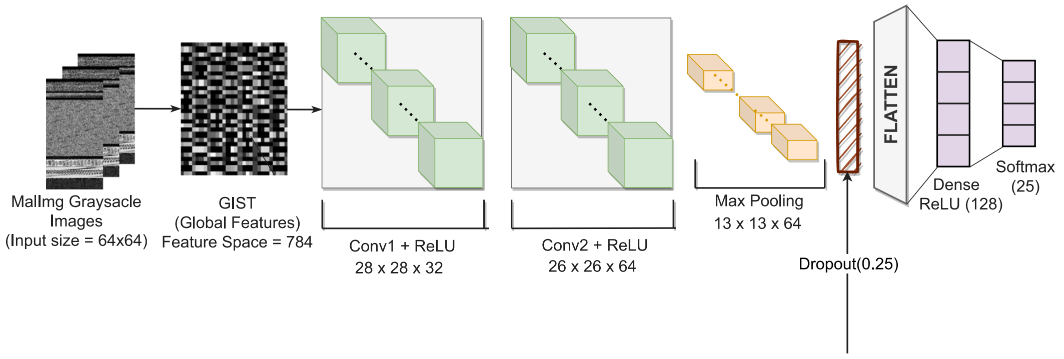


Fig. 6. Structure of proposed CNN integrated with MTIS architecture.

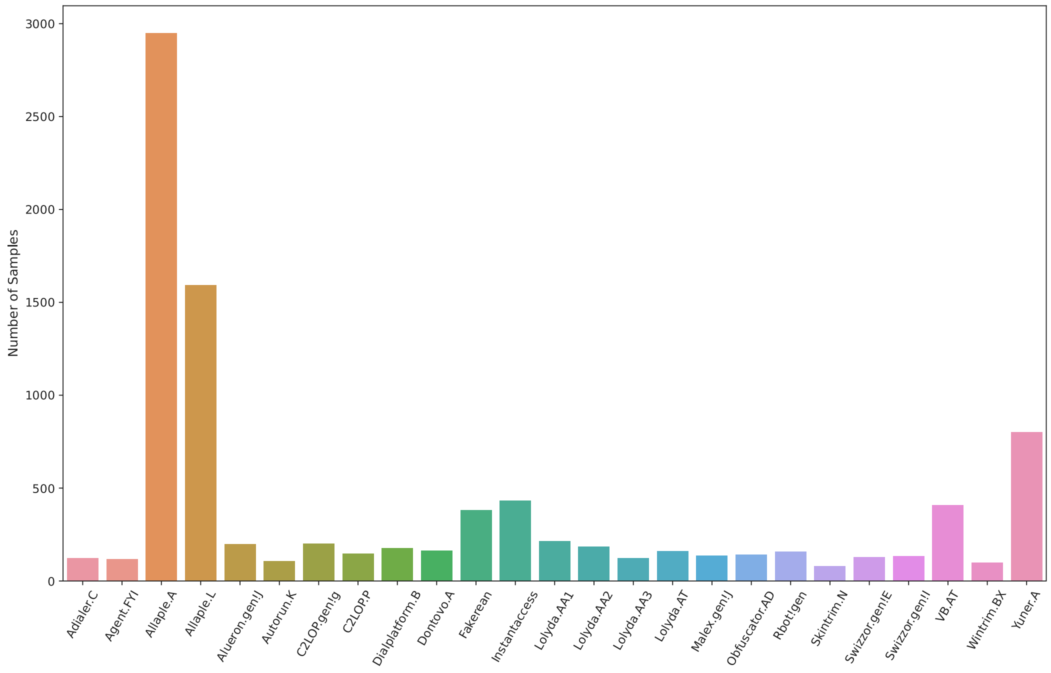


Fig. 7. Distribution of MalImg dataset.

Let E be an error function such that $E_{training\mp loss}(t)$ is the average training error, $E_{val\mp loss}(t)$ is the validation error and $E_{test\mp loss}(t)$ is the test error of the model, calculated after every epoch t.

E_{opt} is defined to be the lowest validation set error obtained in epochs up to t:

$$E_{opt}(t) = \begin{pmatrix} Min \\ t < t' \end{pmatrix} E_{val\mp loss}(t') \tag{10}$$

Further, generalization loss at epoch t is given by

$$GL(t) = 100.\left(\frac{E_{val\mp loss}}{E_{opt}} - 1\right) \tag{11}$$

5. Materials and datasets

This research uses publicly available datasets MalImg [31] (Fig. 7) with 9339 malware samples of 25 distinct families as a benchmark for comparison with other models. Additionally, this study uses real-world samples in the form of Microsoft Window‘s PE files with 690 samples of 22 families (Fig. 8) collected on designed and operationalized MTIS architecture. Over more than one year, malware samples of 240+ families were collected. However, as most malware families have smaller counts, making the CNN model more computationally intensive to deal with them, families with >10 counts are selected for model implementation.

Collected PE files are labeled through a data labeling unit which is further visualized into grayscale images. Since real-world malware datasets are relatively small and imbalanced, data augmentation is applied, which helps tackle this problem by augmenting new and similar instances in the dataset. The augmentation deals with the data scarcity problem and balances it by producing more variability in the data, which reduces the overfitting of the model. Finally, after data augmentation, the CNN model is trained and validated on augmented real data with 2055 samples of 22 families. For MalImg, whole data is split into a 90:10 ratio in experiments, which depicts 90% of the total dataset used for training and 10% used for testing. The real-world malware dataset is divided into a 70:30 ratio to enable the model‘s training. It is assumed that the model will work better to classify real malware once the volume and diversity of data increase.

The research setup for implementation of the proposed methodology is created using a 64-bit Ubuntu 18.04 operating system with Intel Xeon(R)Platinum 8153 CPU@2.00GHz*64, Quadro P400/PCIe/SSE2 graphics card, and 93 GB RAM. The setup used python programming and Keras library at the back-end.

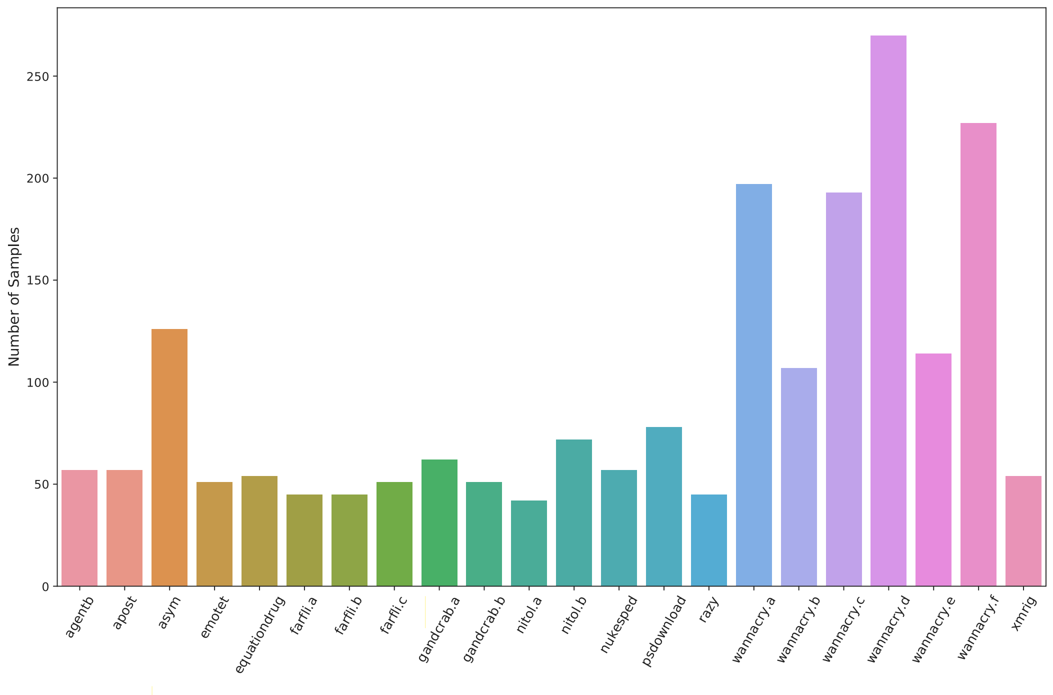


Fig. 8. Distribution of real malware dataset.

Table 5
Accuracy comparison of different algorithms with respect to image resolution.

Image descriptor	Image resolution					
	28 × 28	64 × 64	128 × 128	192 × 192	224 × 224	300 × 300
KNN + DSIFT	82.07	90.20	98.66	**98.34**	98.66	98.34
SVM + DSIFT	86.99	94.00	98.34	**98.50**	98.55	98.45
Random Forest + DSIFT	85.44	91.81	98.13	**98.39**	98.29	98.61
Naive Bayes + DSIFT	72.97	76.18	96.15	**96.52**	96.36	96.36

6. Results

The results are obtained for both datasets — MalImg and real-world malware samples. The experiments are structured as follows: (i) Experiments towards systematic performance comparison of machine learning of MTIS (ML-MTIS) for different image resolution, types of image descriptors (LBP, GLCM, DSIFT, and GIST features), and effect of training and testing data split to obtain the best classifier, (ii) Experiments of deep learning of MTIS(DL-MTIS) with respect to different image resolution and comparative analysis with selected baseline methods i.e., CNN+GIST, CNN, and k-NN+GIST, (iii) Validation results of deep learning CNN method for detection of real-world malware samples, (iv) Results on detection of packed malware with image visualization-based approach (v) Finally, deep learning of MTIS (DL-MTIS), CNN model is compared with the previously studied literature.

6.1. Performance analysis of ML methods of MTIS (ML-MTIS)

In terms of machine learning-based malware classification systems in MTIS architecture, several experiments are performed. First, different image resolution 28 × 28, 64 × 64, 128 × 128, 192 × 192, 224 × 224, and 300 × 300 are used for selection of optimal resolution. Second, the best optimal model concerning accuracy and prediction time is obtained. Third, the effect of training and testing ratio on selected optimal model is observed, enabling selection of the best-fitted model. As mentioned in Section 4.5, this study uses four ML algorithms, namely, k-NN, SVM, NB, and RF, as a baseline comparison.

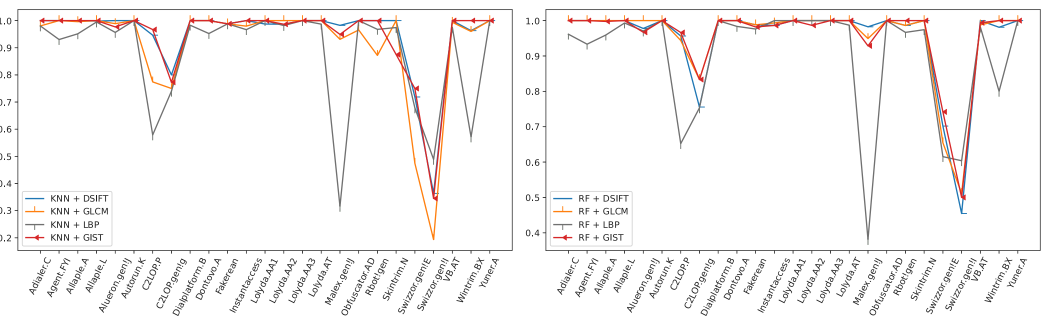
In experiments, a DSIFT descriptor is used to select optimal image resolution because the feature map of DSIFT is more diverse than other local and global feature descriptors. It is noticed that classification

Table 6
Accuracy comparison of different algorithms with respect to feature descriptors.

Feature descriptor		Algorithm			
		k_NN	SVM	NB	RF
LBP	Acc(%)	94.54	79.23	90.52	95.36
	Time(Sec)	0.44	11.86	0.04	1.7
DSIFT	Acc(%)	**98.29**	98.5	96.52	98.23
	Time(Sec)	**9.5**	139.84	65.17	86.36
GLCM	Acc(%)	97.64	83.73	92.29	98.29
	Time(Sec)	2.44	12	0.06	3.32
GIST	Acc(%)	98.07	97.11	97	98.5
	Time(Sec)	3.4	12.97	2.52	17.51
DSIFT+PCA	Acc(%)	98.18	98.29	96.68	**98.55**
	Time(Sec)	2.32	1.62	0.31	**10.56**
GIST+PCA	Acc(%)	98	97.27	96.68	97.7
	Time(Sec)	2.44	4.9	0.28	1.14

accuracy of all four classifiers is almost similar for image size of 128 × 128, 192 × 192, 224 × 224, and 300 × 300 whereas image size 28 × 28, 64 × 64 produced accuracy on lower side. A larger image size requires more computational time for prediction; however, there is a trade-off between accuracy and response time. Moreover, in all the four selected classifiers, an image size of 192 × 192 produced markable accuracy with an optimal response time compared to other image resolutions. Hence, for machine learning classifiers, an image size of 192 × 192 is selected as a baseline for further experiments. Table 5 shows the effect of various image ratios on classification accuracies of the models.

Towards the selection of the best optimal ML classifier for MTIS architecture, experiments are performed using local and global textural



(a) (b)

Fig. 9. F1-score of k-NN and Random forest classifiers.

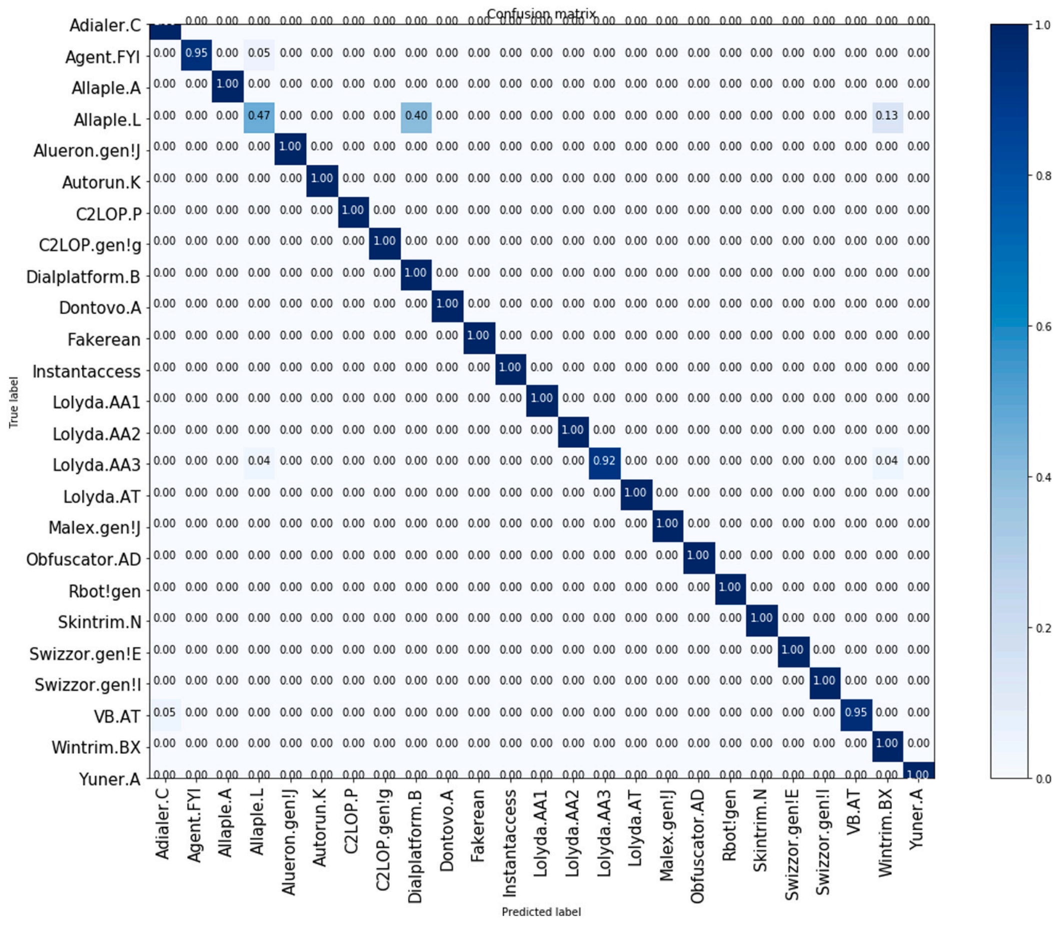


Fig. 10. A confusion matrix of proposed model for MalImg datasets. It indicate the correct classification of Swizzor.gen!E, Swizzor.gen!I which were misclassified by other models.

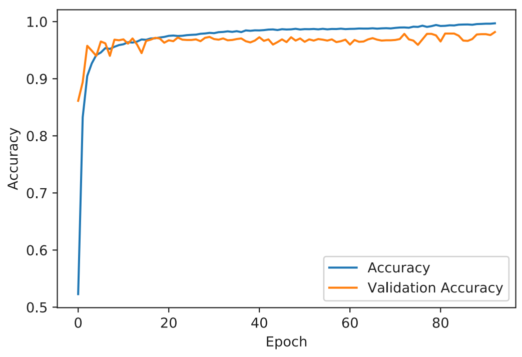
features of an image. Table 6 shows the accuracy and prediction time of all four classifiers for the type of image descriptors. It is observed that in the case of DSIFT, k-NN, SVM, and RF produced a markable accuracy as 98.29%, 98.5%, and 98.23% with k-NN taking the least prediction time of 9.5 s only. Similarly, in the case of DSIFT+PCA(256), the maximum classification accuracy is of the RF classifier at 98.55% and 10.56 s prediction time. It can be concluded that all four classifiers integrated with MTIS architecture performed very well, but RF+DIST+PCA obtained better accuracy with a bearable prediction time. Fig. 9 shows the F1-score comparison of k-NN and RF classifiers corresponding to the type of descriptors. With this experiment, it is concluded that k-NN and RF classifiers produce excellent accuracy in this setup. Hence, these two models are taken into consideration to see the effect of training and testing ratio on classification accuracy.

Table 7 shows a comparative analysis of k-NN and RF classifiers for different data splits. However, the model‘s accuracy varied with the type of descriptors used for feature extraction and training and testing data. Using the LBP descriptor, both classifiers k-NN and RF obtained maximum classification accuracy with a training and testing ratio of 70:30. But for DSIFT, both algorithms obtained maximum accuracy with 80:20 of training and testing data. For GLCM, k-NN achieved maximum accuracy of 97.64% with a 90:10 training and testing data split. In contrast, RF obtained a maximum accuracy of 98.34% with an 80:20 training and testing ratio, but the variation in accuracy is less. Similarly, in the case of the GIST descriptor, k-NN obtained maximum accuracy with 80:20 data split, but there is a very small variation with respect to the ratio of 70:30. RF classifier obtained maximum accuracy of 98.5% with 90:10 data split ratio for GIST descriptor.

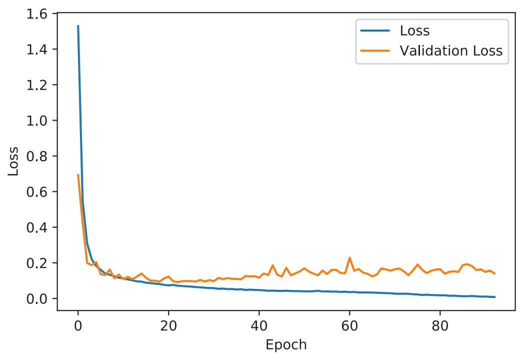
Table 7
Effect of training and testing ratio on performance of k-NN and RF classifiers.

MalImg datasets of Microsoft Windows malware										
	Training Ratio (%)									
Algorithm		10	20	30	40	50	60	70	80	90
KNN+LBP	CA	90.84	92.83	93.21	93.72	93.94	93.66	94.5	94.43	94.54
	F1	88.88	91.66	92.18	92.84	93.08	92.79	93.91	93.82	93.63
	T	0.61	0.62	0.58	0.56	0.55	0.52	0.5	0.46	0.44
KNN+DSIFT	CA	97.01	97.38	97.63	97.75	97.82	98.18	98.25	**98.29**	97.86
	F1	96.66	97.24	97.49	97.61	97.67	98.07	98.15	**98.25**	97.77
	T	5.74	5.76	6.66	6.43	8.27	9.17	9.56	**9.5**	9.41
KNN+DISFT +PCA	CA	97.05	97.46	97.72	97.73	97.84	98.13	98.18	98.18	97.86
	F1	96.73	97.33	97.6	97.59	97.71	98.01	98.07	98.13	97.77
	T	1.06	1.31	1.63	1.89	2.14	2.28	2.32	2.63	2.57
KNN+GLCM	CA	94.21	95.56	96.07	96.2	96.32	96.47	96.79	96.79	97.64
	F1	93.7	95.21	95.77	95.93	96.09	96.24	96.6	96.62	97.49
	T	1.11	1.53	1.87	2.12	2.13	2.45	2.84	2.75	2.44
KNN+GIST	CA	96.84	97.39	97.45	97.36	97.69	97.86	98.04	98.07	97.97
	F1	96.56	97.3	97.34	97.25	97.57	97.8	97.98	98.05	97.81
	T	1.26	1.62	1.92	2.16	2.39	2.52	2.65	3.4	3.67
KNN+GIST +PCA	CA	96.82	97.39	97.46	97.38	97.69	97.83	98	97.97	97.86
	F1	96.55	97.3	97.36	97.27	97.58	97.77	97.95	97.96	97.73
	T	1.02	1.35	1.74	1.86	2.09	2.32	2.44	2.66	2.53
RF+LBP	CA	91.28	94.1	94.66	94.95	94.93	94.49	95.36	95.18	94.75
	F1	90.02	93.34	94.02	94.37	94.27	93.89	94.85	94.68	94.34
	T	0.62	0.77	0.94	1.13	1.31	1.48	1.7	1.83	2.02
RF+DSIFT	CA	96.85	97.58	98.07	97.97	97.88	98.1	98.22	98.23	98.18
	F1	96.54	97.48	97.99	97.89	97.8	98.05	98.19	98.25	98.08
	T	8.84	18.06	27.95	38.86	49.35	60.55	73.16	86.36	98.61
RF+DSIFT +PCA	CA	95.52	96.94	97.49	97.5	97.56	98.02	98.22	**98.55**	97.86
	F1	94.91	96.75	97.37	97.35	97.46	97.94	98.13	**98.46**	97.68
	T	1.27	2.41	3.62	4.86	6.15	7.67	9.27	**10.56**	12.16
RF+GLCM	CA	96.69	97.64	97.8	97.86	97.94	98.07	98.22	98.34	98.39
	F1	96.6	97.56	97.76	97.8	97.88	98.02	98.18	98.34	98.31
	T	0.65	0.9	1.22	1.54	1.87	2.17	2.56	2.93	3.32
RF+GIST	CA	97.28	97.71	97.98	98.16	98.09	98.1	98.04	98.34	98.5
	F1	97.07	97.63	97.92	98.1	98.04	98.06	98.01	98.31	98.32
	T	1.81	3.36	5.18	6.95	8.94	10.92	12.96	15.31	17.51
RF+GIST +PCA	CA	94.71	96.11	97.03	97	96.85	97.08	97.14	97.7	97.32
	F1	93.78	95.69	96.78	96.8	96.63	96.92	96.92	97.65	97.04
	T	1.25	2.31	3.44	4.65	5.96	7.47	8.7	10.14	12.3

Note: CA=Test accuracy, F1=F1-score, T=model prediction time in seconds.



(a)



(b)

Fig. 11. The accuracy and loss graph of the proposed CNN model.

Hence, it can be concluded that data split largely affects the performance of the model. However, there is no optimal split ratio, but it is expected that as the number of training samples increases, the model improves, and at some point, it gets saturated. In the case of the MalImg dataset, it is observed that 933 training samples (10% data) give an adequate generalization on unseen data, almost comparable in performance to 1438 training samples (70% data) of the real-world malware dataset. This fact can also be attributed to biasness and the balanced property of benchmarked datasets, making it easier for the model to gain a learning experience.

6.2. *Performance analysis of DL methods of MTIS (DL-MTIS)*

For a baseline comparison, the proposed CNN model in MTIS architecture is compared with CNN+GIST, CNN with the whole image as an input, and k-NN+GIST. The experiments are performed with a various image ratio of 28 × 28, 64 × 64, 128 × 128, 192 × 192, 224 × 224, and 300 × 300 using GIST descriptor, but it was observed that after image size of 64 × 64, the accuracy of the model saturates and prediction time increases. Hence, image size of 64 × 64 is chosen for CNN model to perform further experiments [31,37]. The grayscale

Table 8
Performance analysis of proposed CNN model with different baseline (Windows malware).

Microsoft Window‘s malware datasets (MalImg)				
Malware family	Proposed method	GIST+CNN	CNN	KNN+GIST
Adialer.C	1.00	1.00	1.00	1.00
Agent.FYI	0.95	0.95	1.00	1.00
Allaple.A	1.00	1.00	1.00	1.00
Allaple.L	0.47	0.47	0.99	1.00
Alueron.gen!J	1.00	1.00	1.00	1.00
Autorun.K	1.00	1.00	1.00	1.00
C2LOP.P	1.00	1.00	0.81	0.85
C2LOP.gen!g	1.00	1.00	0.82	0.29
Dialplatform.B	1.00	0.88	1.00	1.00
Dontovo.A	1.00	1.00	1.00	1.00
Fakerean	1.00	1.00	0.95	1.00
Instantaccess	1.00	1.00	1.00	1.00
Lolyda.AA1	1.00	1.00	1.00	1.00
Lolyda.AA2	1.00	1.00	1.00	1.00
Lolyda.AA3	0.92	0.77	1.00	1.00
Lolyda.AT	1.00	1.00	1.00	1.00
Malex.gen!J	1.00	1.00	0.93	1.00
Obfuscator.AD	1.00	1.00	1.00	1.00
Rbot!gen	1.00	1.00	1.00	1.00
Skintrim.N	1.00	1.00	1.00	1.00
Swizzor.gen!E	1.00	1.00	0.71	0.69
Swizzor.gen!I	1.00	1.00	0.42	0.62
VB.AT	0.95	0.95	0.98	1.00
Wintrim.BX	1.00	0.94	1.00	1.00
Yuner.A	1.00	1.00	1.00	1.00
Test Accuracy	**98.71**	98.07	97.65	97.54

Table 9
Detection of packed malwares.

Family name	Packers	TPR(%)	FPR(%)	Accuracy(%)
agentb	UPX	93.75	0.17	99.68
apost	UPX	100.0	0.0	100.0
equationdrug	UPX,Themida	100.0	0.0	100.0
farfali	UPX,PECompact	86.7901	0.79	98.76
nitol	UPX	85.71	0.17	99.59
Razy	UPX, Themida	92.50	0.35	99.19

malware images are resized to 64*64 for every model. Table 8 shows comparative results of all three deep learning methods. It is observed that CNN+GIST works well on the test dataset with 98.07% accuracy compared to 97.65% of plain CNN and 97.54% of the KNN+GIST method. Further, with early stopping configurations, model accuracy improved to 98.71% on test datasets with 99.5% training accuracy. Besides this, the proposed CNN model has lesser false positives than all other models. Most methods misclassified minor data classes such as Swizzor.gen!E, Swizzor.gen!I, but the proposed model of CNN+GIST with early stopping configuration correctly classifies them. Fig. 10 shows a normalized confusion matrix of CNN+GIST with early stopping configuration. Fig. 11 shows model‘s accuracy and loss graph with early stopping condition of patience q=70. The CNN model was executed for 300 epochs configured with validation loss-based early stopping rules, and the model got saturated at 132th epochs with 99.89% training accuracy and 98.72% validation accuracy.

6.3. Validation with real world malware samples

The proposed CNN model‘s effectiveness and robustness are also tested with real-world malware samples. In the case of real-world malware, the model‘s training and validation accuracy is 97.64% and 93.19%, respectively, at 106th epoch. The model training and validation loss is 0.0645 and 0.3380, respectively, at 106th epoch. The overall average accuracy of the model is 99.4%, and the testing accuracy of the CNN model is seen to be 93.19% with data augmentation and early stopping configurations. The experiment results show excellent performance in classifying real-world malware samples, even with a relatively small data count. The intuition of integrating this CNN model with the attack data capturing framework is that it will automatically learn about new variants of malware families. Moreover, the model will get trained incrementally for unknown and real-world malware samples. It yields a good classifier with accurate labeling, thereby aiding cyber threat intelligence generation without relying on anti-virus technologies.

6.4. Results on detection of packed malware

Code obfuscation is an active research problem in malware detection. Packing or encryption is a favorite tool malware authors use to prevent analysis and thwarting signature-based detection methods. Worse, various packers exist for this purpose, such as UPX, Aspack, NSPack, PECompact, Themida, etc., which gives the easy generation of polymorphic malware. The proposed method using image textural analysis can handle polymorphic code obfuscations such as packing/encryption. Firstly, to evaluate resistance towards detection of obfuscated malware, the proposed method selects 142 malware samples in the Obfuscator.AD malware family of MalImg datasets. The experiment results validated that image textural features-based malware detection is effectively able to classify Obfuscator.AD malware family, and it is resistant to polymorphic obfuscation, i.e., packing/encryption. Further to evaluate it on real-world packed malware, few samples of different families — agentb, equationdrug, farfli, nitol, and razy collected on MTIS architecture, are packed with different packers such as UPX, PECompact, Themida, etc. It is worth mentioning that the proposed method can classify the obfuscated malware packed with these packers. Table 9 shows performance results of these malware families with true positive rate, false-positive rate, and accuracy. Fig. 12 shows a use-case of an equationdrug malware packed with "Themida" packer and classified by DL methods of MTIS.

6.5. Comparison with previously studied methods in literature

A comparison of deep learning CNN methods in recent literature is performed with the MalImg dataset as a benchmark. These existing studies were selected because they used image visualization-based techniques for malware classification. Table 10 shows that the proposed method achieves a better classification accuracy (98.71%), precision (99%), and recall (99%) without domain expertise and feature engineering.

7. Conclusion and future work

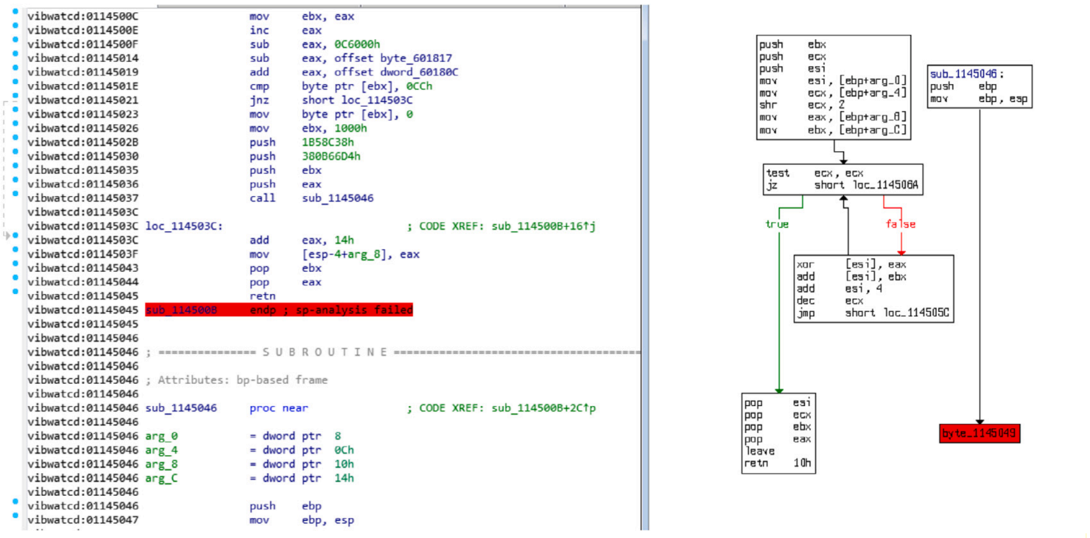
A new malware threat intelligence system(MTIS) architecture is proposed for malware collection and classification that integrates a hybrid approach of ML and DL-based classifications. The Mircosoft Window‘s PE files collected on the MTIS system are systematically labeled into malware families before applying learning algorithms. For labeling, this research leverages AVClass and customized Malheur engines as a combined approach using signature-based and machine learning clustering algorithms to get a correctly labeled dataset. Labeled malware samples are transformed into grayscale images, which convert the malware classification problem into a computer vision problem. In the proposed architecture, both local (LBP, DSIFT, GLCM) and global features(GIST) of a visualized image are used, providing flexibility to use the model based on the type of feature map. A balanced approach to classify malware programs using machine learning and CNN is presented.

Machine learning algorithms, namely, k-NN, SVM, NB, and RF classifiers, are compared corresponding to the image resolution size, training, and testing data split along with the type of features to get the best optimal classifier. The result reveals that image resolution of 192 × 192 is ideal for machine learning-based malware classification, and 64 × 64 is optimal for deep learning methods in MTIS architecture

Table 10
Comparative analysis of proposed method with previously studied ML/DL methods on MalImg dataset.

Method	Year	Dataset	Algo	PR (%)	RR (%)	CA (%)
Deep CNN+weighted softmax [49]	2017	MalImg	DL	–	–	97.18
SVM+Global features [50]	2017	MalImg	DL	–	–	97.32
BAT algorithm+CNN [33]	2018	MalImg	DL	94.6	94.5	94.5
Cui, Zhihua, et al. [33]	2018	MalImg	DL	–	88.40	97.60
DL vs GIST [51]	2018	MalImg	ML	–	–	97
DL method [52]	2018	MalImg	DL	–	–	95.66
M-CNN [53]	2018	MalImg	DL	–	–	98.52
Softmax+CNN [5]	2019	MalImg	DL	–	–	98.186
NSGA-II [52]	2019	MalImg	DL	–	88.4	97.6
MD-IIOT [39]	2020	MalImg	DL	98.47	98.47	–
Proposed method	–	MalImg	DL	99	99	98.71

Note: PR=Precision rate,
RR=Recall rate,
CA=Classification accuracy.



```
vibwatcd:0114500C                  mov     ebx, eax
vibwatcd:0114500E                  inc     eax
vibwatcd:0114500F                  sub     eax, 0C6000h
vibwatcd:01145014                  sub     eax, offset byte_601817
vibwatcd:01145019                  add     eax, offset dword_60180C
vibwatcd:0114501E                  cmp     byte ptr [ebx], 0CCh
vibwatcd:01145021                  jnz     short loc_114503C
vibwatcd:01145023                  mov     byte ptr [ebx], 0
vibwatcd:01145026                  mov     ebx, 1000h
vibwatcd:0114502B                  push    1B58C38h
vibwatcd:01145030                  push    380B66D4h
vibwatcd:01145035                  push    ebx
vibwatcd:01145036                  push    eax
vibwatcd:01145037                  call    sub_1145046
vibwatcd:0114503C
vibwatcd:0114503C loc_114503C:                              ; CODE XREF: sub_114500B+16↑j
vibwatcd:0114503C                  add     eax, 14h
vibwatcd:0114503F                  mov     [esp-4+arg_8], eax
vibwatcd:01145043                  pop     ebx
vibwatcd:01145044                  pop     eax
vibwatcd:01145045                  retn
vibwatcd:01145045 sub_114500B      endp ; sp-analysis failed
vibwatcd:01145045
vibwatcd:01145046
vibwatcd:01145046 ; =============== S U B R O U T I N E =======================================
vibwatcd:01145046
vibwatcd:01145046 ; Attributes: bp-based frame
vibwatcd:01145046
vibwatcd:01145046 sub_1145046      proc near                ; CODE XREF: sub_114500B+2C↑p
vibwatcd:01145046
vibwatcd:01145046 arg_0            = dword ptr  8
vibwatcd:01145046 arg_4            = dword ptr  0Ch
vibwatcd:01145046 arg_8            = dword ptr  10h
vibwatcd:01145046 arg_C            = dword ptr  14h
vibwatcd:01145046
vibwatcd:01145046                  push    ebp
vibwatcd:01145047                  mov     ebp, esp
```

```
push    ebx
push    ecx
push    esi
mov     esi, [ebp+arg_0]
mov     ecx, [ebp+arg_4]
shr     ecx, 2
mov     eax, [ebp+arg_8]
mov     ebx, [ebp+arg_C]

test    ecx, ecx
jz      short loc_114506A

xor     [esi], eax
add     [esi], ebx
add     esi, 4
dec     ecx
jmp     short loc_114505C

pop     esi
pop     ecx
pop     ebx
pop     eax
leave
retn    10h

sub_1145046:
push    ebp
mov     ebp, esp
```

Fig. 12. Analysis ofa equationdrug malware packed with themida packer.

for datasets used in this study. The experiment results show that in the DSIFT descriptor, k-NN, SVM, and RF produce a remarkable accuracy as 98.29%, 98.5%, and 98.23%, but k-NN takes less prediction time as 9.5 s only. Similarly, in the case of DSIFT+PCA(256), the k-NN classifier produces 98.55% test classification accuracy with 10.56 s prediction time. It can be concluded that all four classifiers integrated with MTIS architecture perform well, but k-NN+DSIFT+PCA give better accuracy with a reasonable prediction time. However, the model‘s accuracy varied according to the type of image descriptors used for feature extraction. The results indicate that the proposed CNN model performed better than the selected baseline. Further, with the introduction of early stopping, the model‘s accuracy improved to 98.71% on test datasets. Experiments with a more diverse set of real-world and complex malware samples are proposed as future work. The incubation of varied techniques to detect new malware families and continuous updating of the model using online learning methods is proposed to sustain the model for newer and evolving classes of malware samples.

CRediT authorship contribution statement

Sanjeev Kumar: Data collection, Proof of experiments (PoCs), Coding, Validation, Writing – original draft. **B. Janet:** Conceptualization, Investigation, Methodology, Designing and reviewing, Implementation of the research idea.

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.

References

[1] Cisco, Cisco Annual Internet Report, 2020, https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. (Accessed 4 June 2021).

[2] Symantec, Symantec Threat Landscape Report 2020, 2020, https://symantec-enterprise-blogs.security.com/blogs/feature-stories/symantec-2021-cyber-security-predictions-looking-toward-future. (Accessed 4 June 2021).

[3] M. Egele, T. Scholte, E. Kirda, C. Krügel, A survey on automated dynamic malware-analysis techniques and tools, ACM Comput. Surv. 44 (2008) 6:1–6:42.

[4] K. Kancherla, S. Mukkamala, Image visualization based malware detection, in: 2013 IEEE Symposium on Computational Intelligence in Cyber Security, CICS, 2013, pp. 40–44.

[5] H. Naeem, Detection of malicious activities in internet of things environment based on binary visualization and machine intelligence, Wirel. Pers. Commun. 108 (4) (2019) 2609–2629.

[6] H. Naeem, B. Guo, M.R. Naeem, F. Ullah, H. Aldabbas, M.S. Javed, Identification of malicious code variants based on image visualization, Comput. Electr. Eng. 76 (2019) 225–237.

[7] Y.-s. Liu, Y.-K. Lai, Z.-H. Wang, H.-B. Yan, A new learning approach to malware classification using discriminative feature extraction, IEEE Access 7 (2019) 13015–13023.

[8] K. Sanjeev, B. Janet, R. Eswari, Automated cyber threat intelligence generation from honeypot data, in: Inventive Communication and Computational Technologies, Springer, 2020, pp. 591–598.

[9] S. Kumar, B. Janet, R. Eswari, Multi platform honeypot for generation of cyber threat intelligence, in: 2019 IEEE 9th International Conference on Advanced Computing, IACC, 2019, pp. 25–29.

[10] M.G. Schultz, E. Eskin, F. Zadok, S.J. Stolfo, Data mining methods for detection of new malicious executables, in: Proceedings 2001 IEEE Symposium on Security and Privacy, S&P 2001, IEEE, 2000, pp. 38–49.

[11] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, Y. Elovici, Detecting unknown malicious code by applying classification techniques on opcode patterns, Secur. Inform. 1 (1) (2012) 1–22.

[12] I. Santos, F. Brezo, X. Ugarte-Pedrero, P.G. Bringas, Opcode sequences as representation of executables for data-mining-based unknown malware detection, Inform. Sci. 231 (2013) 64–82.
[13] M. Belaoued, S. Mazouzi, A real-time pe-malware detection system based on chi-square test and pe-file features, in: IFIP International Conference on Computer Science and Its Applications, Springer, 2015, pp. 416–425.
[14] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, E. Bodden, Mining apps for abnormal usage of sensitive data, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 1, IEEE, 2015, pp. 426–436.
[15] R. Lu, Malware detection with lstm using opcode language, 2019, arXiv preprint arXiv:1906.04593.
[16] A.I. Elkhawas, N. Abdelbaki, Malware detection using opcode trigram sequence with SVM, in: 2018 26th International Conference on Software, Telecommunications and Computer Networks, SoftCOM, IEEE, 2018, pp. 1–6.
[17] K. Rieck, P. Trinius, C. Willems, T. Holz, Automatic analysis of malware behavior using machine learning, J. Comput. Secur. 19 (4) (2011) 639–668.
[18] C.-T. Lin, N.-J. Wang, H. Xiao, C. Eckert, Feature selection and extraction for malware classification, J. Inf. Sci. Eng. 31 (3) (2015) 965–992.
[19] S.K. Dash, G. Suarez-Tangil, S.J. Khan, K. Tam, M. Ahmadi, J. Kinder, L. Cavallaro, DroidScribe: Classifying android malware based on runtime behavior, in: 2016 IEEE Security and Privacy Workshops, SPW, 2016, pp. 252–261.
[20] H. Cai, N. Meng, B. Ryder, D. Yao, Droidcat: Effective android malware detection and categorization via app-level profiling, IEEE Trans. Inf. Forensics Secur. 14 (6) (2018) 1455–1470.
[21] A. Pektaş, T. Acarman, Malware classification based on API calls and behaviour analysis, IET Inf. Secur. 12 (2) (2018) 107–117.
[22] M. Belaoued, A. Boukellal, M.A. Koalal, A. Derhab, S. Mazouzi, F.A. Khan, Combined dynamic multi-feature and rule-based behavior for accurate malware detection, Int. J. Distrib. Sens. Netw. 15 (11) (2019) 1550147719889907.
[23] R. Sihwail, K. Omar, K.A. Zainol Ariffin, S. Al Afghani, Malware detection approach based on artifacts in memory image and dynamic analysis, Appl. Sci. 9 (18) (2019) 3680.
[24] M. Ali, S. Shiaeles, G. Bendiab, B. Ghita, MALGRA: Machine learning and N-gram malware feature extraction and detection system, Electronics 9 (11) (2020) 1777.
[25] F.O. Catak, A.F. Yazı, O. Elezaj, J. Ahmed, Deep learning based sequential model for malware analysis using Windows exe API calls, PeerJ Comput. Sci. 6 (2020) e285.
[26] J. Chen, S. Guo, X. Ma, H. Li, J. Guo, M. Chen, Z. Pan, SLAM: A malware detection method based on sliding local attention mechanism, Secur. Commun. Netw. 2020 (2020).
[27] M. Mimura, R. Ito, Applying NLP techniques to malware detection in a practical environment, Int. J. Inf. Secur. (2021) 1–13.
[28] R. Feng, S. Chen, X. Xie, G. Meng, S.-W. Lin, Y. Liu, A performance-sensitive malware detection system using deep learning on mobile devices, IEEE Trans. Inf. Forensics Secur. 16 (2020) 1563–1578.
[29] Y. Dai, H. Li, Y. Qian, R. Yang, M. Zheng, SMASH: A malware detection method based on multi-feature ensemble learning, IEEE Access 7 (2019) 112588–112597.
[30] T. Kim, B. Kang, M. Rho, S. Sezer, E.G. Im, A multimodal deep learning method for android malware detection using various features, IEEE Trans. Inf. Forensics Secur. 14 (3) (2018) 773–788.
[31] L. Nataraj, S. Karthikeyan, G. Jacob, B.S. Manjunath, Malware images: visualization and automatic classification. in: Proceedings of the 8th International Symposium on Visualization for Cyber Security, 2011, pp. 1–7.
[32] B. Narayanan, O. Djaneye-Boundjou, T. Kebede, Performance analysis of machine learning and pattern recognition algorithms for Malware classification, in: 2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS), 2016, pp. 338–342.
[33] Z. Cui, F. Xue, X. Cai, Y. Cao, G.-g. Wang, J. Chen, Detection of malicious code variants based on deep learning, IEEE Trans. Ind. Inf. 14 (7) (2018) 3187–3196.
[34] S. Ni, Q. Qian, R. Zhang, Malware identification using visualization images and deep learning, Comput. Secur. 77 (2018) 871–885.
[35] M. Kalash, M. Rochan, N. Mohammed, N.D. Bruce, Y. Wang, F. Iqbal, Malware classification with deep convolutional neural networks, in: 2018 9th IFIP International Conference on New Technologies, Mobility and Security, NTMS, IEEE, 2018, pp. 1–5.
[36] E. Rezende, G. Ruppert, T. Carvalho, A. Theophilo, F. Ramos, P. de Geus, Malicious software classification using VGG16 deep neural network's bottleneck features, in: Information Technology-New Generations, Springer, 2018, pp. 51–59.
[37] V.S.P. Davuluru, B.N. Narayanan, E.J. Balster, Convolutional neural networks as classification tools and feature extractors for distinguishing malware programs, in: 2019 IEEE National Aerospace and Electronics Conference, NAECON, IEEE, 2019, pp. 273–278.
[38] Y. Zhao, C. Xu, B. Bo, Y. Feng, Maldeep: A deep learning classification framework against malware variants based on texture visualization, Secur. Commun. Netw. 2019 (2019).
[39] H. Naeem, F. Ullah, M.R. Naeem, S. Khalid, D. Vasan, S. Jabbar, S. Saeed, Malware detection in industrial internet of things based on hybrid image visualization and deep learning model, Ad Hoc Netw. 105 (2020) 102154.
[40] Z. Cui, L. Du, P. Wang, X. Cai, W. Zhang, Malicious code detection based on CNNs and multi-objective algorithm, J. Parallel Distrib. Comput. 129 (2019) 50–58.
[41] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, Q. Zheng, IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture, Comput. Netw. 171 (2020) 107138.
[42] Y. Ding, X. Zhang, J. Hu, W. Xu, Android malware detection method based on bytecode image, J. Ambient Intell. Humaniz. Comput. (2020) 1–10.
[43] X. Huang, L. Ma, W. Yang, Y. Zhong, A method for windows malware detection based on deep learning, J. Signal Process. Syst. 93 (2) (2021) 265–273.
[44] F.O. Catak, J. Ahmed, K. Sahinbas, Z.H. Khand, Data augmentation based malware detection using convolutional neural networks, PeerJ Comput. Sci. 7 (2021) e346.
[45] G. Sun, Q. Qian, Deep learning and visualization for identifying malware families, IEEE Trans. Dependable Secure Comput. (2018).
[46] L. Van der Maaten, G. Hinton, Visualizing data using t-SNE, J. Mach. Learn. Res. 9 (11) (2008).
[47] M. Sebastián, R. Rivera, P. Kotzias, J. Caballero, Avclass: A tool for massive malware labeling, in: International Symposium on Research in Attacks, Intrusions, and Defenses, Springer, 2016, pp. 230–253.
[48] L. Prechelt, Automatic early stopping using cross validation: quantifying the criteria, Neural Netw. 11 (4) (1998) 761–767.
[49] S. Yue, Imbalanced malware images classification: a CNN based approach, 2017, ArXiv abs/1708.08042.
[50] A. Makandar, A. Patrot, Malware class recognition using image processing techniques, in: 2017 International Conference on Data Management, Analytics and Innovation, ICDMAI, 2017, pp. 76–80.
[51] S. Yajamanam, V.R.S. Selvin, F. Di Troia, M. Stamp, Deep learning versus gist descriptors for image-based malware classification, in: Icissp, 2018, pp. 553–561.
[52] N. Bhodia, P. Prajapati, F. Di Troia, M. Stamp, Transfer learning for image-based malware classification, 2019, arXiv preprint arXiv:1903.11551.
[53] M. Kalash, M. Rochan, N. Mohammed, N.D.B. Bruce, Y. Wang, F. Iqbal, Malware classification with deep convolutional neural networks, in: 2018 9th IFIP International Conference on New Technologies, Mobility and Security, NTMS, 2018, pp. 1–5.

Sanjeev Kumar has obtained his Bachelor of Technology (B. Tech) in Computer Engineering and Master of Technology (M. Tech) in Computer Science and Engineering. He has a vast research and development experience in network security, malware analysis, machine learning, honeypots, and honeynets. He has worked on the latest cybersecurity technology, such as the active honeypot system for malicious URL detection, botnet detection, cyber-attack data capturing, and collections through Honeypots/Honeynets, cyber threat intelligence generation. He has guided more than 14 intern students in the completion of their project and thesis work. He has established a state-of-the-art distributed honeypot clusters at various geo-locations covering various regions/sectors for malware capturing, collections, and cyber threat intelligence generations.

B. Janet is an Assistant Professor at the Department of Computer Applications, NIT, Trichy. She obtained B.Sc. (Physics) with Distinction from Holy Cross College, Trichy affiliated to the Bharathidasan University, Tiruchirappalli, Obtained a Master of Computer Applications from Bishop Heber College, with the third rank from Bharathidasan University, Tiruchirappalli. She cleared the National Eligibility Test (NET) for Lectureship by UGC in 2001. She obtained Master of Philosophy in Computer Science from Alagappa University, Karaikudi and Ph. D. degree from National Institute of Technology, Tiruchirappalli. Her area of research is Information Retrieval, deep learning, information security. She has published research papers in International Journals and Conferences of repute in Information retrieval and security. She is also a reviewer for many International Conferences. She is presently a Life Member of the Computer Society of India, International Association of Engineers, ISTE, ISC, and International Association of Computer Science and Information Technology. She is also an active member of IEEE and ACM.