Available online at www.sciencedirect.com

ScienceDirect

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

Computers & Security

Malware detection employed by visualization and deep neural network

Anson Pinhero[a], Anupama M L[a], Vinod P[b,*], C.A. Visaggio[c,*], Aneesh N[a], Abhijith S[a], AnanthaKrishnan S[a]

[a] Department of Computer Science and Engineering, SCMS School of Engineering and Technology, Cochin, India
[b] Department of Computer Applications, Cochin University of Science and Technology, Cochin, India
[c] Department of Engineering, University of Sannio, Benevento, Italy

ARTICLE INFO

Article history:
Received 7 August 2020
Revised 23 December 2020
Accepted 22 February 2021
Available online 26 February 2021

Keywords:
Malware classification
Malware detection
Malware visualization
Machine learning
Deep learning

ABSTRACT

With the fast growth of malware's volume circulating in the wild, to obtain a timely and correct classification is increasingly difficult. Traditional approaches to automatic classification suffer from some limitations. The first one concerns the feature extraction: static approaches are hindered by code obfuscation techniques, while dynamic approaches are time consuming and evasion techniques often impede the correct execution of the code. The second limitation regards the building of the prediction models: the adequateness of a training dataset may degrade over time or can not be sufficient for some malware families or instances. With this paper we investigate the effectiveness of a new approach that uses malware visualization, for overcoming the problems related to the features selection and extraction, along with deep learning classification, whose performances are less sensitive to a small dataset than machine learning. The experiments carried out on twelve different neural network architectures and with a dataset of 20,199 malware, demonstrate that the proposed approach is successful as produced an F-measure of 99.97%.

1. Introduction

According to the 2019 McAfee Labs Threat Report, there are about 70M new malwares circulating in the wild, while the cumulative number of malicious software accounts for 1 Billion of instances[1]. Such a volume calls for an effective automatic detection and classification of malware. The scenario is made more severe if we consider the 25% rise in the number of attacks due to destructive malware, i.e. malware that is able to damage hardware components, as reported by the Symantec Internet Security Threat Report 2019 (Symantec, 2019).

Traditionally malware detection was signature-based. Even though the signature based pattern matching is a quick process, it is successful if the malware is known, i.e. the signature is correctly extracted and stored within the antivirus database; otherwise, if the malware was never detected or the hash cannot be extracted because the code is obfuscated or

* Corresponding authors.
E-mail addresses: anson.pinhero@scmsgroup.org (A. Pinhero), anupama.ml@scmsgroup.org (A. M L), vinod.p@cusat.ac.in (V. P), visaggio@unisannio.it (C.A. Visaggio), aneesh.n@scmsgroup.org (A. N), abhijith.s@scmsgroup.org (A. S), ananthakrishnan.s@scmsgroup.org (A. S).

[1] https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-aug-2019.pdf

https://doi.org/10.1016/j.cose.2021.102247
0167-4048/

ciphered, the detection based on the signature fails. Additionally, the signatures database requires a large amount of storage, given the huge volume of malware to observe.

Malware analysis is essential for identifying the most relevant behaviors - like the mechanisms characterizing the payload, the spreading dynamics, the hiding techniques, the lateral movements, the boot survival - of the malicious software, and assign them to one or more specific families. Malware analysis methods can be divided into two approaches: static and dynamic. Static analysis (Chan and Yiu, 2012; David et al., 2016; Huang et al., 2013; Schultz et al., 2001; Zhongyang et al., 2013) examines the code by disassembling the malware binary file without executing it. Different structural patterns are extracted from the binary file, such as strings, hashes, byte sequence n-grams, opcode frequency distribution, permissions, and intents. For performing static analysis, the malware executable has to be unpacked and decrypted in advance. Dynamic analysis (Bläsing et al., 2010; Lin et al., 2018; Qiao et al., 2014; Shabtai et al., 2010; Yan and Yin, 2012) requires the malware to be executed in a controlled and isolated environment (virtual machine or sandbox). Despite the fact that static analysis does not consume a significant time for execution, it is usually vulnerable to code obfuscation like packing, encryption, and code transposition. On the contrary, dynamic analysis is not affected by code obfuscation techniques, but it is time consuming, computationally intensive, and could fail to exhibit interesting behaviours of the program, since the relevant execution paths could be not activated.

In machine learning based malware detection techniques (Li et al., 2018; Liu et al., 2016; Shijo and Salim, 2015; Wang et al., 2015; Wu and Hung, 2014), relevant features such as permissions, intents, API calls, system calls are extracted upfront and used for training classifiers that will generate the models used for classifying the samples of the testing dataset. Classification based malware detection has many advantages like it doesn't require lots of labeled data, it could detect unknown malware, has flexibility to choose a blend of approaches. However, it could sometimes lead to wrong predictions due to inadequate training of the model. Moreover, feature engineering and selecting important attributes is a critical task. Both the feature engineering and the attributes representation require a deep knowledge and proper expertise. To surpass these limitations, deep learning based models (Akarsh et al., 2019; David and Netanyahu, 2015; He and Kim, 2019; Hu and Tan, 2017; Jang et al., 2020; Jun-ling and Shuo-hao, 2019; Kim et al., 2019; Kumar and Khan, 2018; Mitsuhashi and Shinagawa, 2020; Narayanan and Davuluru, 2020; Raff et al., 2017; Ren et al., 2019; Tobiyama et al., 2016; Yan et al., 2018; Yuan et al., 2014; 2016; Zhao et al., 2019) have been developed. As a matter of fact, neural networks are capable of extracting relevant features automatically. Furthermore, a strong domain specific knowledge is not required.

Malware visualization is a technique that enables humans to visually analyze the features in malware. In prior works, authors have visualized binaries as grayscale images (Nataraj et al., 2011), byteplots (Kancherla and Mukkamala, 2013), image matrices (K. S. Han, 2013), images and entropy graphs (Han et al., 2015). Other techniques based on visualization have been: self organizing maps to visualize virus (Yoo, 2004), dynamic analysis to visually depict overall flow of a program (Quist and Liebrock, 2009), visualization of the output produced by different malware detection tools (Goodall et al., 2010), fast hashing methods have used image processing techniques for malware clustering (Arefkhani and Soryani, 2015). An image based malware classifier is agnostic with respect to the type of file, it helps to visually represent distinguishing features of malware that may make it different from a benign file, and to catch the difference between different families of malware with less domain specific knowledge. In most of the cases the visualized images are given as input to the neural network in order to build the decision models for the detectors. Even though, there are a lot of malware visualization techniques, but challenges still remain like real time malware detection, low detection accuracy, delayed detection of new malware which includes zero-day exploits.

In our study, we compare three kinds of image for malware visualization: grayscale, RGB and Markov images. While grayscale and RGB images differ for the compression mechanism, thus entail a different quantity and quality of the original information about the malware, a Markov image reduces the dimensionality of the representation.

Markov models are known for their applications in solving challenging problems like temporal pattern recognition based on sequential data like speech, handwriting, and bioinformatics. Markov model captures the statistical properties of a sequence of symbols that comprise a pattern. Previous research (Abbas Alipour and Ansari, 2019; Salehi and Amini, 2017; Tajoddin and Jalili, 2018; Xiao et al., 2017; Zhang and Xiao, 2017) has shown that Markov model based malware detection produces better results. Markov models here are used to obtain images of malware that owns only one dimension.

Gabor filter is a linear filter and it is used in image processing for representing points where texture changes within an image. To analyze the images' texture is useful for different purposes such as features extraction for image retrieval and classification of pixels and regions around a pixel. Many contemporary vision scientists claim that frequency and orientation representations of Gabor filters are similar to those of the human visual system (Olshausen and Field, 1996). Gabor filters allow to extract features of the spectrum which are spatially localized. This is the reason why we investigate its application for comparing malware's images.

In this paper, we investigate a method for malware detection and classification based on malware visualization and deep learning. We conducted a set of experiments using 12,971 benign samples and two malware datasets: Microsoft Malware Classification challenge BIG 2015 (Ronen et al., 2018), including 10,860 labelled samples belonging to 9 malware families and Malimg dataset (Nataraj et al., 2011), including 9,339 samples belonging to 25 malware families. The "Desktop Operating System Market Share Worldwide - May 2020" Report (StatCounter Global Stats, 2020) shows that Windows has the 77.02% market share: this is the reason why our approach focused on the detection of Windows executables. Initially, we visualized the input files as grayscale, RGB and Markov (Abbas Alipour and Ansari, 2019; Salehi and Amini, 2017; Tajoddin and Jalili, 2018; Xiao et al., 2017; Zhang and Xiao, 2017) images. Finally, we performed texture analysis using Gabor filter (Grigorescu et al., 2002; Turner, 1986) on all these images. We also varied the image dimensions (32×32,

64 × 64, 128 × 128 and 256 × 256). All the four types of images were then given as input to twelve different neural network architectures for classification. The validation of the proposed method proposed evidence about the effectiveness of Entropy images (Han et al., 2015) for multi-class classification to different malware families. Specifically, some models are strongly effective, as we obtained an F-measure varying in the range 82.34% to 99.97%. In summary, the main contributions of our work are:

1. a novel method for grayscale and RGB image generation using colormap;
2. the application of Gabor filter to all the three types of images (grayscale, RGB and Markov image) for extracting relevant features;
3. 232 models using the above mentioned four types of images;
4. the experimentation considers as variation factors the image dimension and neural network architectures: models were generated using twelve different neural network architectures including VGG3 (Simonyan and Zisserman, 2015) and ResNet50 (He et al., 2016) and 4 different image dimensions.

The rest of the paper is arranged as follows. In Section 2, a thorough study about the state-of-the-art malware detection techniques is mentioned. In Section 3 a detailed description about the proposed framework is provided. The experiments and the results are described in Section 4. Section 5 discusses the results thus obtained. A comparative study of our work with existing models is done in Section 6. Finally the paper is concluded in Section 7, along with future scope.

2. Related work

In order to highlight the novelty of our work, we examine the state of the art regarding three main topics the proposed method is related to: the techniques for determining when two malware instances can be declared similar, malware detection and classification through machine learning and deep learning algorithms.

2.1. *Similarity based malware detection techniques*

Zhong et al. (2012) created a feature database based on the analysis of known malware programs. In the case of unknown malware, these functions are compared to the content of the database to determine the family it belongs to. Authors use a filtering algorithm based on one-class SVM to calculate similarity . Finally, they carried out an experiment with 113 malware samples. They observed that their approach consumes less time for similarity calculation. B. Kang *et al.* Kang et al. (2012) propose a malware classification method, that is based on block comparison , and identifies the core parts of binaries that can represent a family of malware, and thus reduce the overhead of using the whole file.

Andro-profiler (Jang et al., 2016) is a behavior-based mobile malware detector. It executes malicious application on emulator to extract integrated system logs. Then, generates behavior profiles by analyzing the system logs. It compares the behavior profile of malicious application with representative behavior profiles for each malware family using a weighted similarity matching technique. Other authors proposed (Taheri et al., 2019) four malware detection techniques using Hamming distance which are first nearest neighbors (FNN), all nearest neighbors (ANN), weighted all nearest neighbors (WANN), and k-medoid based nearest neighbors (KMNN). They evaluated these techniques on three datasets with benign and malware Android apps like Drebin, Contagio and Genome. They tested their models on features like API, intent and permissions of the three datasets. Using API as features, they achieved higher accuracy compared to other state of art methods.

2.2. *Machine learning based malware detection techniques*

Nataraja et al. (2010) proposed a method for detecting packed executables. It analyzes only raw binaries ; bigrams are extracted and used for training an SVM based classifier. The experimental results show that it is able to detect packed executables with a high detection rate. Crowdroid (Burguera et al., 2011) is an Android malware detector that uses dynamic analysis; it allowed to find that the trojanized applications launched more system calls than benign applications. In Crowdroid, the authors have implemented a simple two-means clustering algorithm to distinguish malware and benign apps. In order to collect traces from a large number of users they introduced their framework based on crowdsourcing. It is a lightweight application that alerts the users when downloading malware and can isolate malware.

The model proposed in (Patel and Buddhadev, 2015) extracts static features from the manifest file of the application using `aapt` command. Additionally, it extracts function calls by executing the application in Android emulator. Subs Later, feature selection is performed on the basis of the information gain. Finally, a rule generation module is used to create rules which map permissions with function calls. The model achieved a detection rate of 96.4%. However, it has limitations like high power and storage consumption. Chuang and Wang (2015) proposed a machine learning model for malware detection. Initially, the input applications are disassembled. Then the count of each API call used by the application is obtained. A similar statistics is obtained for both benign and malicious apps. They observed that the count of malicious API is higher in malicious apps compared to benign. Using SVM training on two different feature sets (malicious and normal), they built a classifier model to improve the accuracy. Achievement of this model is that, it can be used to make decisions on labeling unknown applications.

2.3. *Deep learning based malware detection techniques*

Rezende et al. (2017) have proposed a Deep Convolutional Neural Network (DCNN)model for malware family classification. They developed the model by representing the samples as byteplot images and by applying the transfer learning approach. MCSC (Malware Classification using SimHash and CNN) Le (2018) is another malware family classification model

which transforms disassembled malware codes into gray images. Techniques like multi-hash, major block selection and bilinear interpolation are applied.

Malware analysis of imaged binary samples by using CNN with attention mechanism is performed in Yakura et al. (2018). The proposed method calculates the attention map which depicts regions having higher importance for classification. Qiao et al. (2019) proposed a multi-channel visualization method for malware classification based on deep learning. Initially, malware binary file is converted into a 256 × 256 matrix. Secondly, a 256-dimensional vector of each byte in binary file is obtained to form a 256 × 256 matrix. Subsequently, another 256-dimensional vector of each assembly instruction in each file is computed, and one more 256 × 256 matrix is generated. Finally, threematri ces are visualized into an uncompressed multi-channel image. Then it is fed to LeNet5 for classification. Summary of existing malware detection models is as given in Table 1. In this research paper, we conduct a comprehensive analysis for detecting malware, and malware is classified by representing binaries in the form of images. In particular, we performed our investigation using both gr ay-scale and colour images of variable dimensions using two benchmark dataset, i.e., Malimg and Microsoft Big 2015. We performed an in-depth analysis by implementing several CNN models (both customized and transfer learning approaches). Additionally, we evaluated the performance of the classifier on Markov images (using pixel probabilities for training the deep networks), besides we extracted the texture (using Gabor filter) from images to evaluate the performance of models. Finally, we extended our work by classifying malicious files into its respective families by training the model using images derived from block entropy computed over segments of executables. The overall process investigates detection and classification mechanisms that have not been studied in literature.

3. The proposed framework

Our framework consists of five phases: data collection, data preprocessing, model generation, model training and samples classification. After having collected goodware and malware samples, four types of images were generated. The hexadecimal representation of every file's binary is converted into an image and given as an input to a neural network, which is trained with relevant textures, features of the image and classifies the file as either benign or malware. The detailed architecture of the proposed system is illustrated in Fig. 1.

3.1. *Data collection*

The dataset used in the experimentation was made of malware and benign executable files. Malware executables used in this experiment were obtained from two datasets, namely Malimg (Nataraj et al., 2011) and Microsoft Malware classification challenge (Big 2015) (Ronen et al., 2018). Benign executables were collected from various sources and analyzed with the VirusTotal service(VirusTotal, 2020), which uses 70 anti-malware systems for establishing whether a file is malicious or not.

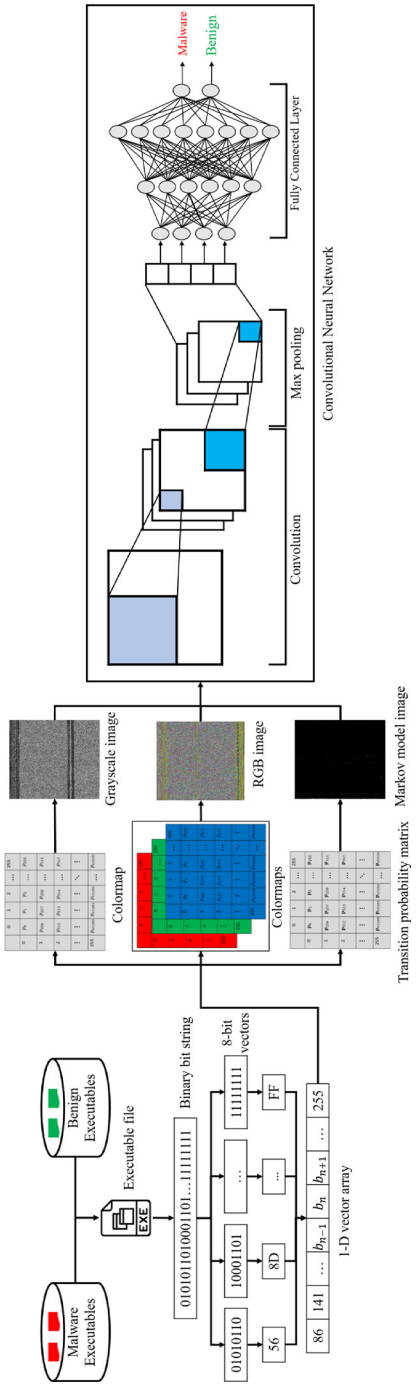


Fig. 1 – Proposed system architecture.

Table 1 – Summary of existing malware detection models.

Author	Description
Nataraja et al. (2010)	• A technique to detect packed executable. • Based on bigram-based features. • Involves the use of SVM for training and testing. • Obtained detection rate in the range of 95%-98%.
Nataraj et al. (2011)	• Malware binaries are visualized as grayscale images. • GIST is leveraged to compute texture features. • Malware family classification is performed using KNN. • Dataset is Malimg with 9339 samples. • Achieved 98.08% classification accuracy.
Kang et al. (2012)	• MBC (Major Block Comparison) system. • Based on signatures extracted from malware binary. • Helps to reduce computational overheads.
K. S. Han (2013)	• Malware binary is extracted through static analysis. • Malware binary is visualized as image matrices. • Similarities are computed among different malware. • It can be easily automated to analyze malware.
Pascanu et al. (2015)	• Malware classification is done by using ESN and RNNs. • Combined recurrent model with standard classifiers. • Found that ESN models outperform RNNs.
Makandar and Patrot (2015)	• Malware is visualized as grayscale images. • Based on Gabor wavelet transform and GIST. • Feed-forward Artificial Neural Network is used. • Dataset is Mahenhur with 3131 malware samples. • Malware classified with 96.35% accuracy.
Saxe and Berlin (2015)	• A deployable malware detector using static features. • First component extracts features from binaries. • Second component is DNN classifier. • Final component is score calibrator. • Achieved 95% detection rate at 0.1% FPR.
Han et al. (2015)	• Malware family classification using binary files. • Malware is visualized as images and entropy graphs. • Malware is classified by calculating similarities. • Small false-positive/false-negative rate.
Vinayakumar and Soman (2018)	Shallow and deep networks are evaluated. • LR, NB, KNN, DT, RF, SVM, RBF and DNN are used. • DNN exhibited high results.
Kalash et al. (2018)	• Malware family classification. • Malware binaries are visualized as grayscale images. • A deep CNN architecture for classification is designed.
Sun and Qian (2018)	• Malware family classification using feature image. • Combined static analysis with RNN and CNN. • Minhash is used to generate feature image. • Experimented on BIG 2015 malware dataset.
Cui et al. (2018)	• Malware classification using CNN. • A solution was proposed for data imbalance problem. • Experimented on Malimg dataset. • Obtained 94.5% accuracy.
Le et al. (2018)	• Malware classification using BIG 2015 dataset. • One-dimensional representation of binary file. • Obtained an accuracy of 98.2% using deep learning.
Rezende et al. (2018)	• Malware family classification using VGG16. • Experimented on 10136 samples from 20 families. • Achieved an accuracy of 92.97%.
Agarap and Pepito (2018)	• Malware classification using deep learning and SVM. • Dataset Malimg with 9339 samples. • CNN-SVM, GRU-SVM and MLP-SVM were applied. • GRU-SVM with 5 layers performs better. • It has an accuracy of 84.93 %.
Yakura et al. (2018)	• Malware visualized as imaged binary samples. • Uses CNN with attention mechanism. • It can process packed malware.

(*continued on next page*)

Table 1 (*continued*)

Author	Description
Le (2018)	• Malware Classification using SimHash and CNN. • Used disassembled malware code. • Malware is visualized as gray images. • Finally classified using CNN.
Venkatraman et al. (2019)	• A hybrid malware classification model. • Uses both supervised and unsupervised learning. • Visualizes malware as grayscale images. • Utilizes both machine learning and deep learning.
Qiao et al. (2019)	• A multi-channel malware visualization method. • Generates three 256 × 256 matrixes of malware. • Combined the three matrixes into a multi-channel image. • It uses LetNet5 for classification.
Shiva Darshan and Jaidhar (2019)	• Windows malware detector using CNN. • Employs API calls and their category as features. • Images are built by using N-gram files. • Experimented on 3282 benign and 4151 malware files. • Resulted in an accuracy of 97.968%.

Malimg dataset (Nataraj et al., 2011) contains 9,339 malware samples from 25 different families. The dataset is created by collecting the samples from Anubis analysis system (Anubis, 0000). Each executable in this dataset is represented in the form of a grayscale image, generated by plotting individual byte values of executable as pixels. Since our proposed approach utilizes colourmap for generating images, we reversed the grayscale image into a hexadecimal representation. Further, we used the hexadecimal values to index the colourmap, where the intersection of the row and column index identifies a pixel.

The Microsoft Malware classification challenge (Big 2015) (Ronen et al., 2018) dataset contains 10,860 labelled malware files from 9 different families. Each file is associated to raw data containing the hexadecimal representation of the file's binary content and metadata information extracted from the binary, such as function calls, strings, and opcode were provided. The metadata was generated using the IDA pro disassembler tool (IDA pro, 0000). Since the experiment aims at investigating image visualization as a technique for detecting malware, only the raw data, i.e. the hexadecimal representation of file's binary content, is considered for the further processing.

Benign dataset contains 12,971 harmless executables. Those executables were collected from software market places: Softonic (Softonic, 2020), Sourceforge (Sourceforge, 2020), Portable Freeware (Portable freeware collection, 2020) and Driverpack Solution (Driverpack Solution, 2020). Since the executables come from diverse sources, the samples needed to be verified as benign samples and duplicate samples had to be removed. Duplicate samples were removed by comparing the SHA-256 hash of each file. In order to verify that all of those collected samples were benign, each file was uploaded to VirusTotal (VirusTotal, 2020) and those samples which were labelled as benign by 100% of the anti-virus engines in the VirusTotal report were selected and only those were included in the benign dataset.

3.2. *Data preprocessing*

All the binary executable files have been split into 8-bits sub strings; the binary content of each string has been mapped to a decimal value in the range from 0 to 255. By doing so, each binary executable has been converted into a 1-D vector of decimal numbers.

3.3. *Image generation*

3.3.1. *Grayscale image*

Fig. 2 illustrates the process of a grayscale image generation from the 1-D vector . For plotting a pixel, two consecutive decimal values from the 1-D array are required. A pixel in the grayscale image is plotted by using a 2-D colourmap, by using a random number generator. We use decimal values 0 and 1 as row and column indices respectively. The resulting 1-D pixel array is reshaped to a 2D-matrix and visualized as a grayscale image. The chosen width of the image was 512 pixels and height is kept variable as it depends directly on the file size.

3.3.2. *RGB image*

A pixel in the RGB image is specified by the amount of red, green, and blue colours. To obtain the contribution of red, green, and blue in a pixel, three different colormaps for red, green, and blue colours are used. For plotting a pixel in the RGB image, two consecutive values from the 1-D vector array are used as row and column indices. The resulting 1-D pixel array is reshaped to a 2-D matrix and visualized as an RGB image. Each image has a fixed width of 512 pixels and a variable height. However, experiments can be performed on variable width, as discussed in Nataraj et al. (2011). The process involved in the RGB image generation is shown in Fig. 3.

3.3.3. *Markov image*

In our solution, we use Markov model to generate image based on 1-D vector array obtained from the binary executable file. The process is illustrated in Fig. 4.

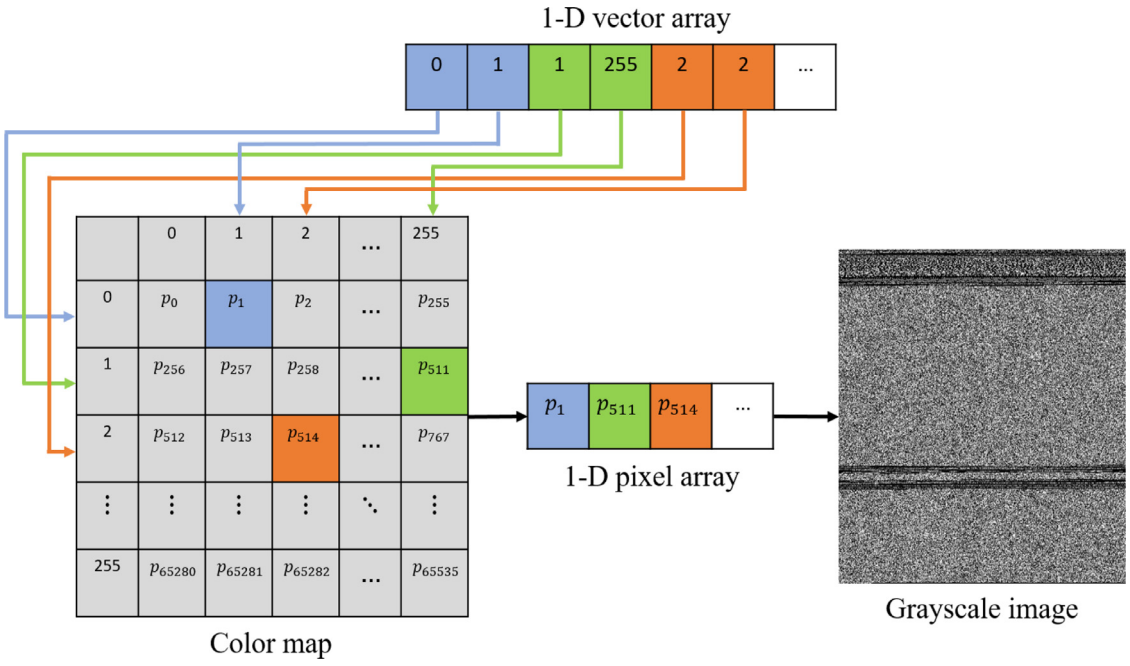


Fig. 2 – Generation of grayscale image.

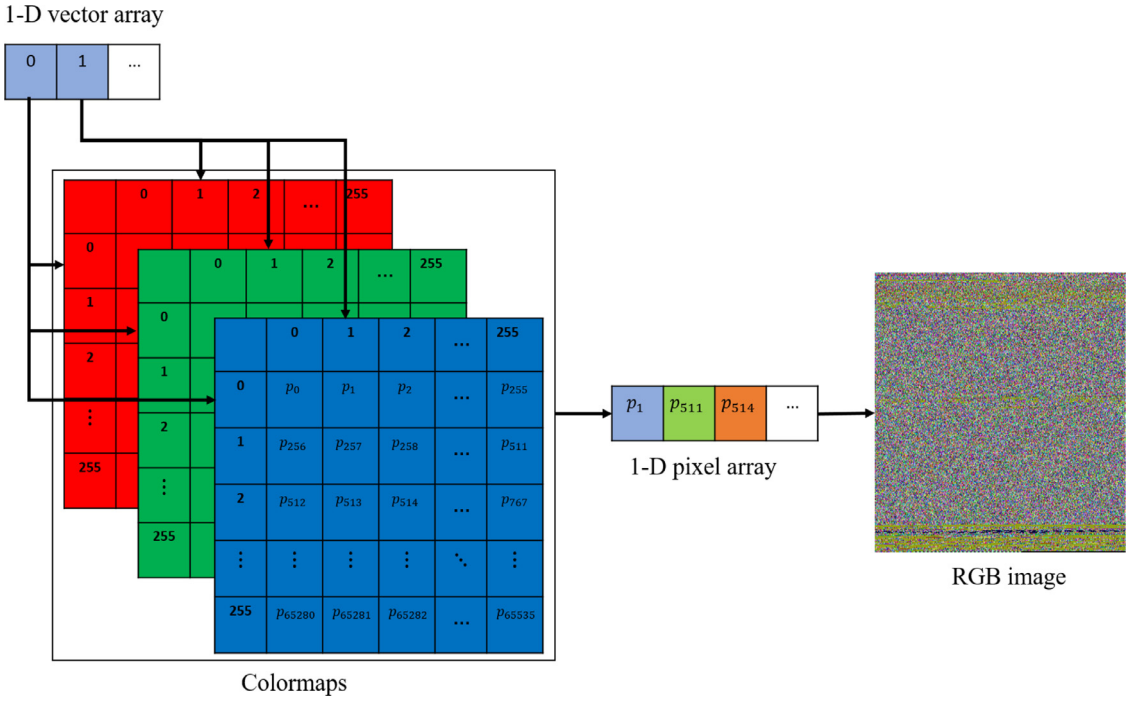


Fig. 3 – Generation of a RGB image.

The generation of Markov image is provided by Algorithm 1. The input to the algorithm is the set of the byte values extracted from a binary executable file. From line number 6 to 12, we determine the frequency of occurrence of byte b_i followed by b_{i+1} and b_i followed by b_k where $0 \leq k \leq 255$. Lines from 15 to 21 compute the probability that byte b_i is followed by b_{i+1}. Finally, lines 24 to 29 compute pixels in Markov image. The mathematical formulation of a pixel in Markov model based images is represented by the following equation:

$$P = \left(TM[i][j] * \left(\frac{255}{MP}\right)\right) mod\ 256 \quad (1)$$

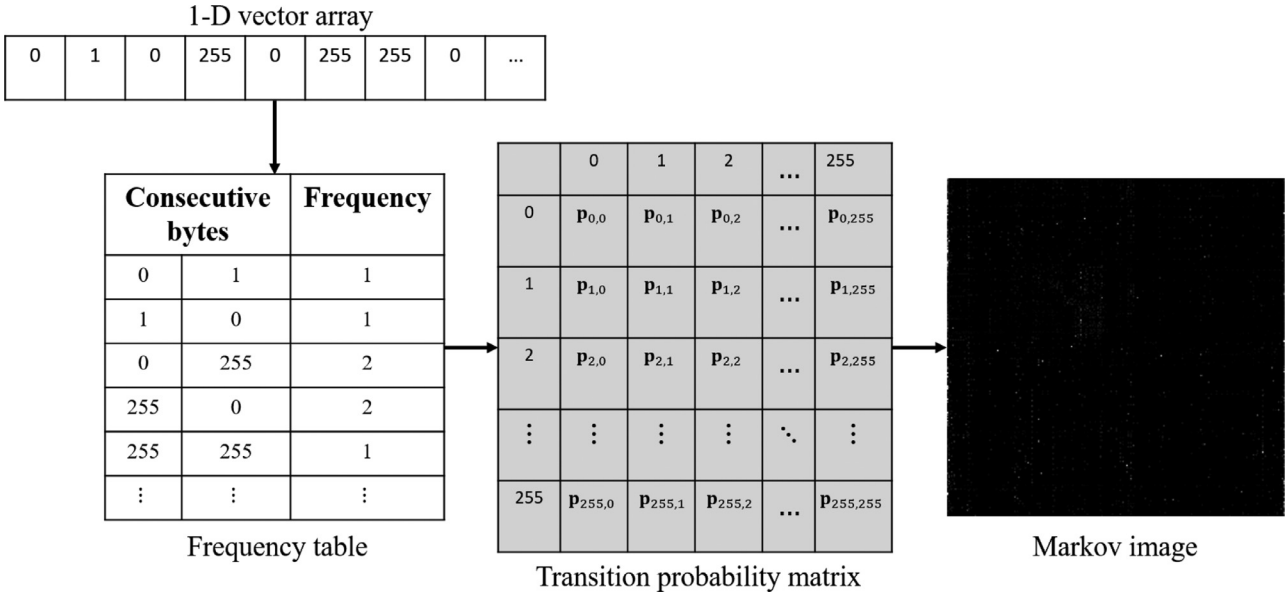


Fig. 4 – Generation of a Markov image.

Algorithm 1 The Generation of a Markov image.

Input: $B = \{b_1, b_2, b_3 \ldots b_n\}$ is a set where b_i represents the decimal value of a byte.

Output: $M = \{m_1, m_2, m_3 \ldots m_n\}$ is a set where m_i represents a pixel value in Markov image.

Initialize TM, 256×256 matrix with zeroes;
Initialize S, 256×1 matrix with zeroes;
$MAX(x)$ is a function to get the largest element from set x.
$LENGTH(x)$ is a function to get number of elements in the set x.
$L = LENGTH(B)$;
$i = 0$;
while $(i < L - 1)$ **do**
    $r = B[i]$;
    $c = B[i + 1]$;
    $TM[r][c] = TM[r][c] + 1$;
    $S[r] = S[r] + 1$;
end while
$i = 0$;
$j = 0$;
while $(i < 256)$ **do**
    $rs = S[i]$;
    while $(j < 256)$ **do**
        $TM[i][j] = TM[i][j]/rs$;
    end while
end while
$MP = MAX(TM)$;
$i = 0$;
$j = 0$;
while $(i < 256)$ **do**
    while $(j < 256)$ **do**
        $p = ((TM[i][j] * 255)/MP) mod\ 256$;
        $M[i][j] = p$;
    end while
end while

Where

- $TM[i][j]$ represents the probability that byte b_i is followed by b_j.
- MP is the maximum probability obtained from the transition probability matrix TM.

3.3.4. *Gabor filter for generating feature image*

In our framework, finally images are generated by using a Gabor filter (Grigorescu et al., 2002; Turner, 1986) to carry out texture analysis. Gabor filter is a feature extractor which represents points where texture changes. It is a sinusoidal signal of particular frequency and orientation. A Gabor filter is represented by the following equation:

$$g(x, y; \sigma; \theta; \lambda; \gamma; \psi) = exp\left[-\frac{x^2 + \gamma^2 y^2}{2\sigma^2}\right].exp\left[i\left(2\pi\frac{x}{\lambda} + \psi\right)\right] \quad (2)$$

- Kernel size (x,y) indicates the type and range of values that each pixel in the Gabor kernel can hold. Here, its value is 3.
- Standard deviation (σ) controls the overall size of the Gabor envelope. Here, its value is 3.
- Theta (θ) indicates the orientation of the Gabor function. Here, its value is 180.
- Wavelength (λ) indicates the width of the strips of Gabor function. Here, its value is 180.
- Aspect ratio (γ) indicates the height of the Gabor function. Here, its value is 0.5.
- Phase offset (ψ) defines the symmetry. Here, its value is 0.

Through experimentation, images created using the above values results in files depicting visual similarities within a family. Thus, the Gabor filter is generated with once fixed these values (Fig. 5).

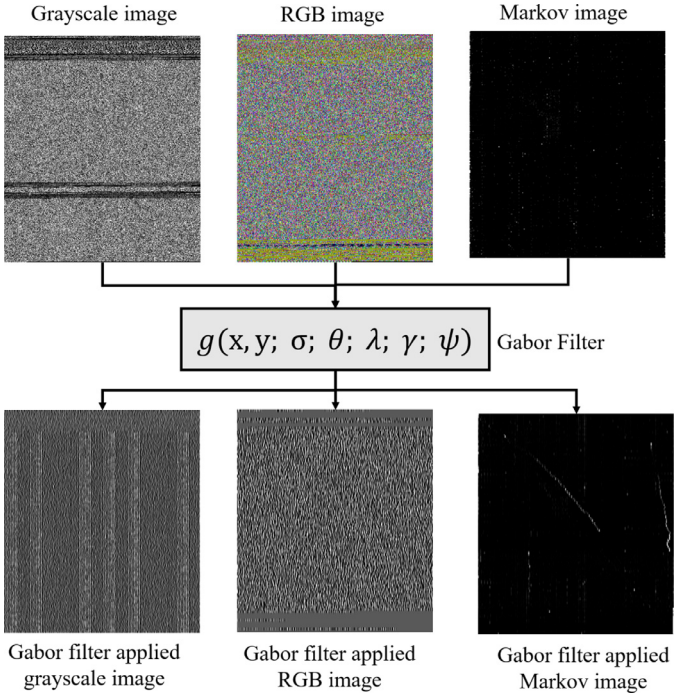


Fig. 5 – Generation of a Gabor filter applied feature image.

3.4. *Convolutional neural network*

The traditional machine learning approach could be ineffective in malware classification, since the training set could become easily not representative of the malware population as discussed in Pendlebury et al. (2019). As a matter of fact, over time the training set could become obsolete or significant families are not enough represented in the training. For this reason, traditional machine learning requires huge data volume for an adequate training and this increases the effort of the analyst necessary to align the classifiers to the needs posed by the variety of malware landscape. A deep learning approach could help to face such a complexity while improving the overall performances of the classification, since it can learn the significant features on an unstructured and unlabelled dataset (Lecun et al., 1998).

Convolutional Neural Network (CNN) is a deep learning approach that have proven very effective in tackling problems such as image recognition and classification (Krizhevsky et al., 2012). CNN consists of a number of convolutional and subsampling layers optionally followed by fully connected layers. Convolution layer captures high-level features from the input image as depth increases, whereas the subsampling layer reduces the spatial size to decrease the number of parameters.

VGG3 is the visual geometry group and includes three stages. VGG is one of the advanced architecture of CNN (Simonyan and Zisserman, 2015). VGG is a stronger and deeper network than the usual convolutional network. It contains almost 138M parameters mostly on the fully connected layers. There are several types of VGG's, the difference on these types is by the number of layers used in it. VGG3 is used in the proposed method for the user resource availability including the storage.

Each stage contains two convolution layers and a pooling layer. The size of the convolution layer used is 3×3 and the pooling layer is 1×1. Using a 3×3 filter allows for expressing more information about the image across all the channels while keeping the size of the convolution layer consistent with the size of the image. The filter size of the pooling layer is 1×1 in the VGG network because the VGG incorporates 1×1 pooling layers to make the decision function more non-linear without changing the receptive fields. The small size of convolution filters allows VGG to have a large number of weight layers which leads to improved performance.

For the experimental purpose, different types of VGG architectures have been used by including batch normalization and dropout layers respectively. So, the different models consist of VGG3 baseline; that is the basic VGG architecture without adding any dropout or batch normalization layer, and the other two models of VGG3 with dropout and VGG3 with dropout and batch normalization. Residual network (Resnet) (He et al., 2016) is one of the most powerful deep neural networks: it works on skip connections or jumps over some layers. By using this skip connection it can avoid vanishing gradients by using activations from a previous layer until the adjacent layer learns its weight.

ResNet-50 is the residual network with fifty layers. It contains four stages, input image is fed into the dimension of multiples of 32. The kernel size of convolution layer used in ResNet is 7×7 and for the pooling layer it is 3×3. The four stages contain several convolution and identity blocks. Each block has three layers. In the initial case the newly computed value will either become smaller or eventually vanish. To save the gradient value identity mapping is used. By that the gradient can directly pass through the gradient gateway and they do not have to pass any weight layers. Hence the values of

Table 2 – Different CNN architectures used in the proposed system.

Model	Architecture	Best-hyperparameter
M-1	Conv + Conv + Pool + Dense +Dense	Epoch=50, LR=0.000075
M-2	Conv + Pool + Dense + Dense	Epoch=50, LR=0.00001
M-3	Conv + Conv + Conv + Pool + Dense + Dense	Epoch=200, LR=0.01
M-4	Conv + Pool + Conv + Dense + Dense	Epoch=120, LR=0.01
M-5	Conv + Globalpool + Dense + Dense	Epoch=50, LR=0.01
M-6	Conv + Conv + Globalpool + Dense + Dense	Epoch=50, LR=0.01
M-7	Conv + Pool + Conv + Pool + Dense + Dense	Epoch=250, LR=0.01
M-8	Conv + Pool + Conv + Pool + Conv + Pool + Dense + Dense	Epoch=200, LR=0.01
M-9	**VGG3 Baseline** - Conv + Conv + Pool + Conv + Conv + Pool + Conv + Conv + Pool + Dense + Dense + Dense + Dense	Epochs=30, LR=0.0001
M-10	**VGG3 Baseline with Dropout** - Conv + Conv + Pool + Dropout + Conv + Conv + Pool + Dropout + Conv + Conv + Pool + Dropout + Dense + Dropout + Dense + Dropout + Dense + Dropout + Dense	Epochs=30, LR=0.0001
M-11	**VGG3 Baseline with Dropout and Batch Normalization** - Conv + Batchnorm + Conv + Batchnorm + Pool + Dropout + Conv + Batchnorm + Conv + Batchnorm + Pool + Dropout + Conv + Batchnorm +Conv + Batchnorm + Pool + Dropout + Dense + Batchnorm + Dropout + Dense + Batchnorm + Dropout+ Dense + Batchnorm + Dropout + Dense	Epochs=30, LR=0.0001,
M-12	**ResNet-50** - ResNet-50 + Dense + Dense + Dense + Dense	Epochs=30, LR=0.0001

the computed gradients cannot be changed. After processing the four stages, there is an average pooling layer followed by a fully connected layer that is used as the final stage of the Resnet and output is fetched from the fully connected layer.

Table 2 shows the twelve different CNN architectures used in the proposed system. These different architectures are applied to four different image dimensions like 32×32, 64×64, 128×128 and 256×256. While experimenting on these different dimensions hyper-parameters have been changed to generate better results. Optimizer used in all the models is SGD with momentum as 0.9. These models are executed on different numbers of epochs with changing learning rate that is mentioned in Table 2. Filter size of convolution layer used is 3×3 for all the models. The usage of different number of neurons and dropout values helped to analyze the behaviour of result generation.

4. Experimental evaluation

This section describes the experimental setup and results obtained with the experiments carried out to generate 232 deep learning models, using different architectures specified in Table 2. It includes 96 models generated using the grayscale image with different image dimensions, ten models based on RGB images, eight models generated using Markov images, 114 models using Gabor filter applied on grayscale, RGB, Markov images, and four models based on entropy images.

4.1. *Dataset description*

We built two datasets for the experiment, namely Dataset-I and Dataset-II. Dataset-I contains 23,831 samples consisting of 10,860 malware from Microsoft Malware Classification Challenge - BIG 2015 (Ronen et al., 2018) and 12,971 goodware executables. Additionally, we created Dataset 2 with 22,310 samples; out of which 9,339 are malware examples taken from Malimg (Nataraj et al., 2011) dataset and 12,971 goodware executables. In both the datasets the goodware was selected with the process previously described. Each dataset was split into two sub-sets, where 70% of the samples were used for training, and 30% of the samples were reserved for prediction.

The programs implementing the experimental process were written in Python using Keras-2.3.1 and TensorFlow-2.2.0 as backend. The hardware configuration of the platform for executing the classification consisted of: an Intel(R) Xeon(R) CPU @ 2.20GHz *2, a 12GB RAM, with an Nvidia Tesla P100 16GB. Since CNN requires a fixed size input image, all the images generated need to be resized to a fixed size before being inputted to the network. Therefore images were resized to different dimensions: 32×32, 64×64, 128×128 and 256×256 pixels, by using an image interpolation technique called INTER_AREA, which is a resampling technique using pixel area relation. The experiments carried out are as listed below:

1. Model generation using grayscale image on Dataset-I and Dataset-II with 32×32, 64×64, 128×128 and 256×256 image dimensions.
2. Model generation using RGB image on Dataset-I and Dataset2 with 256×256 image dimension.
3. Model generation using Markov image on Dataset-I and Dataset-II with 256×256 image dimension.
4. Model generation using gabor filter applied grayscale image on Dataset-I and Dataset-II with 32×32, 64×64, 128×128 and 256×256 image dimensions.
5. Model generation using gabor filter applied Markov image on Dataset-I and Dataset-II with 256×256 image dimension.
6. Model generation using gabor filter applied RGB image on Dataset-I and Dataset-II with 256×256 image dimension.
7. Classification of malware executables by using entropy images, and heterogeneous feature space including block entropy, histograms of pixels and statistical attributes.

4.2. Evaluation metrics

The performance of the classifiers were evaluated with the metrics traditionally used in machine learning literature: accuracy, F-measure, precision and recall. Malware classified as malware is True Positive (TP), malware classified as benign is False Negative (FN), goodware classified as malware is False Positive (FP), while goodware classified as goodware is True Negative (TN). Accuracy is the fraction of samples predicted correctly. Precision is the fraction of predicted positive events that are actually positive. Recall is the fraction of positive events that are predicted correctly. F-measure is the harmonic mean of precision and recall.

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

$$Precision(P) = \frac{TP}{TP + FP} \tag{4}$$

$$Recall(R) = \frac{TP}{TP + FN} \tag{5}$$

$$F - measure(F) = 2 * \left(\frac{P * R}{P + R}\right) \tag{6}$$

4.3. Experimental results

4.3.1. Performance analysis of models using grayscale images

Fig. 6 shows the results obtained for the experiments performed for different grayscale image dimensions on Dataset-I and Fig. 7 on Dataset-II.

From Fig. 6a, it can be observed that on Dataset-I for image dimension 32 × 32, the model M-3 (three convolution layers followed by a pooling and a fully connected layer) obtained 94.10% accuracy and 94.09% F-measure while model M-10 (VGG3 with dropout) shows poor performance: 85.52% of accuracy and 85.49% of F-measure respectively. In M-3, initially, each input image has to go through three convolution layers of 32 neurons each. Then it has to go through a max pooling layer and finally through a fully connected layer of 16384 neurons. It executed for 200 epochs, with a batch size of 32, and a learning rate of 0.0001. As shown in Fig. 7a on Dataset-II for image dimension 32 × 32, the highest performing model M-11 (VGG3 with dropout and batch normalization) has an increase in accuracy by 4.14% compared to the best performing model M-3 (three convolution layers followed by a pooling and a fully connected layer) on Dataset-I. Its architecture is composed of VGG3 followed by three fully connected layers of 4096 neurons each. In this model dropout rates of 0.2, 0.2, 0.3, and batch normalization were included between the convolution-pooling layers of VGG3. In addition, same dropout rates and batch normalization were added in between fully connected layers. It can be noted that, compared to the other models M-11 (VGG3 with dropout and batch normalization) has used both dropout and batch-normalization as the regularization techniques which resulted in its better performance compared to other models. The Model M-12 (ResNet50) has poor performance: 87.16% of accuracy and 83.30% of F-measure.

As for 32 × 32 on Dataset-II, the model M-11 (VGG3 with dropout and batch normalization) has higher performance on both the datasets. Figs. 6b and 7 b show an increase in accuracy by 3.97% compared with the performance of M-11 (VGG3 with dropout and batch normalization) on Dataset-I to Dataset-II. Similarly, the poor performing model is M-5 (convolution layer followed by a global pool and a fully connected layer). In M-5 (convolution layer followed by a global pool and a fully connected layer), global max pooling summarizes the entire values in a feature map to a single value. In M-11 (VGG3 with dropout and batch normalization) the dropout rates of 0.2, 0.3, 0.4 and batch normalization assigned between the convolution-pooling layers of VGG3, also the same added between the three fully connected layers of 4096 neurons each has increased the accuracy by 3.23% and F-measure by 3.27% compared to the model M-9 (VGG3 Baseline). This shows that the regularization techniques contribute to better results. The hyperparameters for M-11 (VGG3 with dropout and batch normalization) were 30 epochs, batch-size of 64 and a learning rate of 0.0001 on Dataset-I while the number of epochs were increased to 200 for M-11 (VGG3 with dropout and batch normalization) on Dataset-II.

For the image dimension 128×128 on Dataset-I, M-11 (VGG3 with dropout and batch normalization) obtained the best performances with dropout rates as 0.1, 02, 0.3 in between the VGG-3 convolution-pooling layers and three fully connected layers on executing it for 30 epochs. M-5 (convolution followed by global pool and a fully connected layer) has again poor performances. The Figs. 6c and 7 c reports the performance measures obtained for image dimension 128×128 on Dataset-I and Dataset-II. The model M-12 (ResNet50) is the top performing model on Dataset-II, with accuracy of 96.47% and F-measure of 96.48%. Its network topology includes ResNet-50 along with three fully connected layers of 1024, 512 and 256 neurons. The model M-5 (convolution layer followed by a global pool and a fully connected layer) obtained poor performances on Dataset-II.

Among the twelve architectures experimented with image dimension 256 × 256 on Dataset-I, the model M-10 (VGG3 with dropout) with dropout rates 0.1, 0.2, 0.3 exhibited the highest performance. It can be noted that dropout, a regularization technique where randomly selected neurons are dropped-out during training, is effective in generalizing the network and reducing the overfitting of trained data. Here, in the case of M-10 (VGG3 with dropout) the network topology is VGG3 followed by three fully connected layers with 1024, 2048 and 4096 respectively. Hyperparameters are leraning rate of 0.0001 and batch size of 64. Identical trends of 64×64 and 128×128 are followed by 256 × 256 as the model M-5 (convolution layer followed by a global pool and a fully connected layer) with global pooling has wrost performances. It is depicted in Fig. 6d. On Dataset-II the highest accuracy and F-measure of 99.46% is achieved by model M-11 (VGG3 with dropout and batch normalization) with dropout rates 0.1, 0.2, 0.3. It took 150 epochs for execution and learning rate of 0.001. The Fig. 7d indicates that here the poor accuracy and F-measure of 93.12% and 93.13% is exhibited by model M-8 (three convolution and pooling layers followed by a fully connected layer).

4.3.2. Performance analysis of models using RGB image

We observed improved results, with identical experiments carried out on the RGB image, for 32 × 32, 64 × 64, 128 × 128,

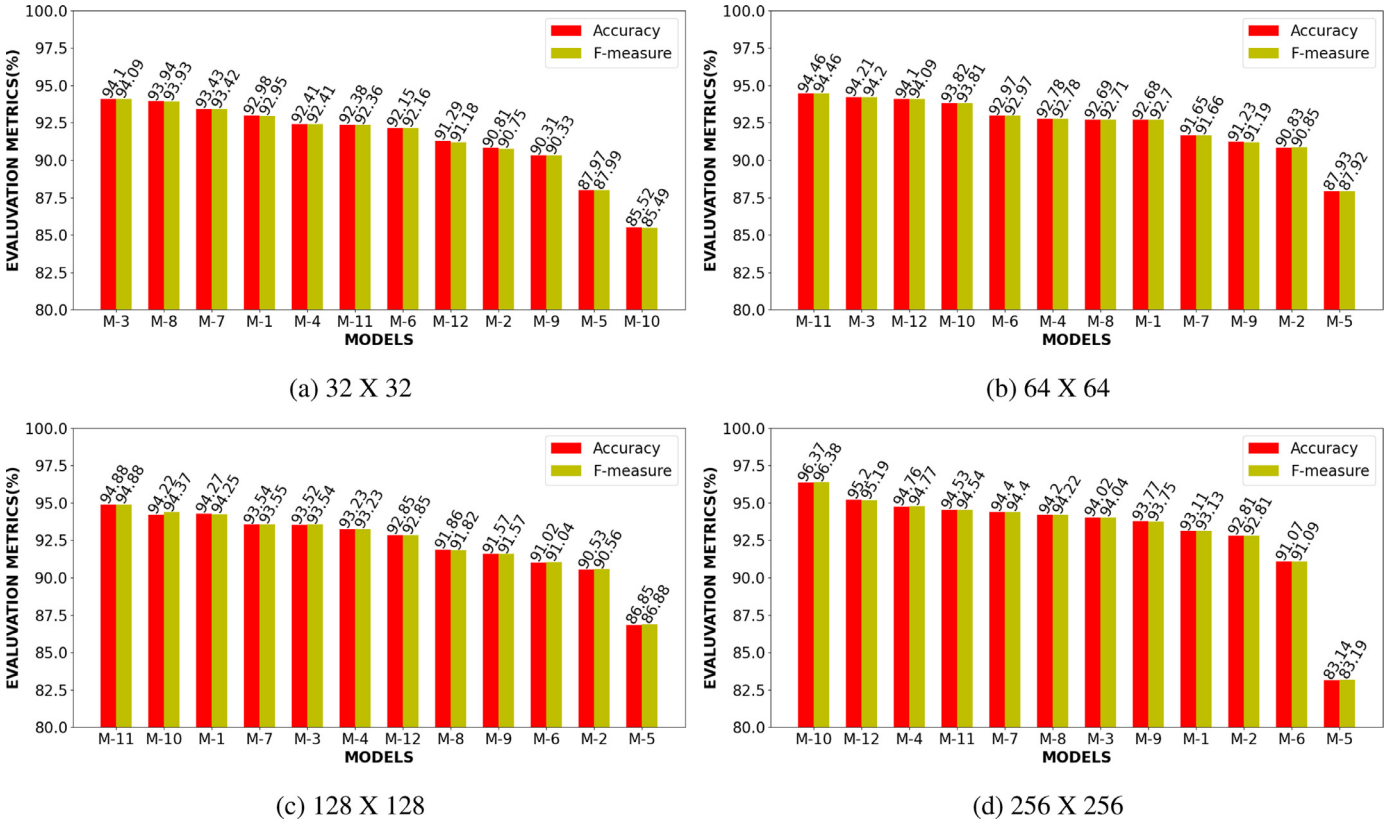
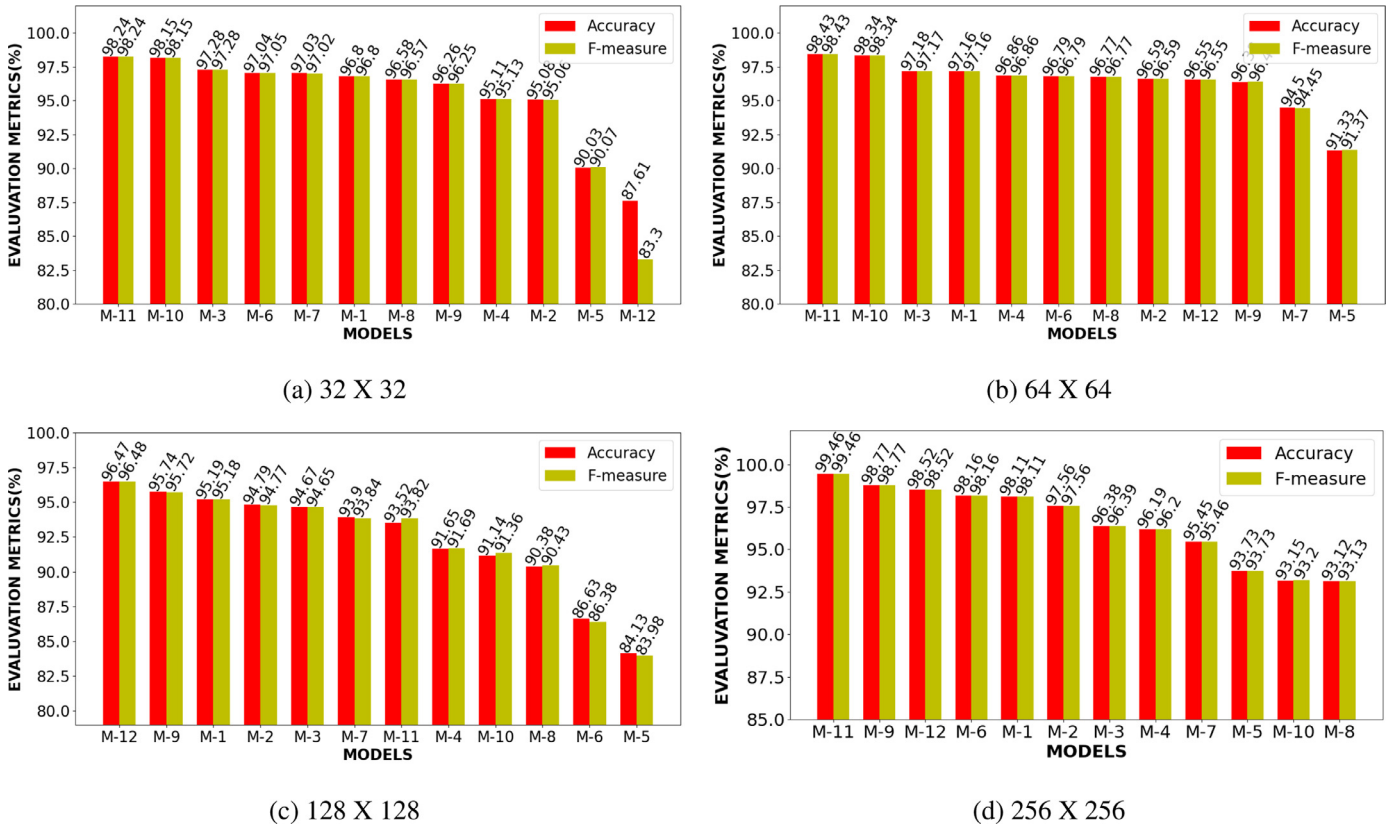


Fig. 6 – The performance of models generated using grayscale image on Dataset-I (Benign and BIG 2015 malware).



(a) 32 X 32 (b) 64 X 64

(c) 128 X 128 (d) 256 X 256

Fig. 7 – Performance of models generated using grayscale image on Dataset-II (Benign and Malimg malware).

Table 3 – The results of models generated using RGB images with dimension 256 x 256.

Dataset	Model	A (%)	F (%)
Dataset-I	M-9	97.38	97.38
	M-10	97.27	97.27
	M-11	97.06	97.07
	M-12	97.06	97.06
	M-4	96.60	96.60
Dataset-II	M-9	99.21	99.21
	M-12	97.87	97.87
	M-10	97.76	97.76
	M-11	97.33	97.33
	M-6	94.18	94.18

Table 4 – The results of models generated using Markov images with dimension 256 x 256.

Dataset	Model	A (%)	F (%)
Dataset-I	M-11	97.97	97.96
	M-12	96.62	96.63
	M-10	93.74	93.76
	M-9	93.72	93.73
Dataset-II	M-11	99.16	99.16
	M-12	97.25	97.25
	M-10	95.76	95.77
	M-9	95.50	95.51

and 256×256 image dimensions. In this section, we discuss the best output, obtained using image dimension 256 × 256 on different models as specified in Table 3 on both the datasets. The best performing model is M-9 (VGG3 baseline) executed for 30 epochs with learning rate of 0.0001. Comparing the results of RGB with Grayscale for image dimension 256 × 256 on Dataset-I, for model M-9 (VGG-3 baseline) there is an increase of 3.6% in accuracy, for model M-11 (VGG3 with dropout and Batch Normalization) it is 2.5%, in the case of model M-12 (ResNet50) it is 1.9% and finally for model M-4 (convolution and pooling followed by convolution and fully connected layer) an increase of 1.8%. On Dataset-II, the increase falls in the range 0.4% to 5%.

4.3.3. *Performance analysis of models using Markov images*

In order to investigate, whether the probability of pixels helps in malware versus benign classification, we performed similar experiments on Markov images as discussed in Section-III (Proposed Framework). Table 4 reports the best results. For both the datasets, model M-11 (VGG3 with dropout and batch normalization) reports high performance measures with a rise of 0.89% and 1.83% in F-measure compared to the model M-11 (VGG3 with dropout and batch normalization) generated using RGB images of Dataset-I and Dataset-II respectively. The model M-11 (VGG3 with dropout and batch normalization) was executed for 50 epochs with dropout rates 0.1, 0.2, 0.3 and learning rate 0.0001 on Dataset-I. For Dataset-II the dropouts were 0.1, 0.2, 0.3 with a learning rate of 0.000007. However, the epochs remained the same.

4.3.4. *Performance analysis of models using Gabor filter applied images*

Since many malware variants are generated by reusing the malicious piece of code, texture analysis methods can visualize these hidden codes and thus help neural networks to differentiate malware samples from benign ones. In order to effectively detect malware, as each malware family uses a specific behaviour to infect the system, exhaustive experiments were carried out on Gabor filter applied images generated in four different dimensions 32 × 32, 64 × 64, 128 × 128 and 256×256 using both the datasets. Further classification models were presented with several Gabor images created by varying parameters such as *kernel size, standard deviation(σ), theta(θ), wavelength(λ), and aspect ratio(γ)*. Detailed analysis was conducted for different combinations of parameter values to obtain set of Gabor images that exhibited high degree of separation amongst malware families.

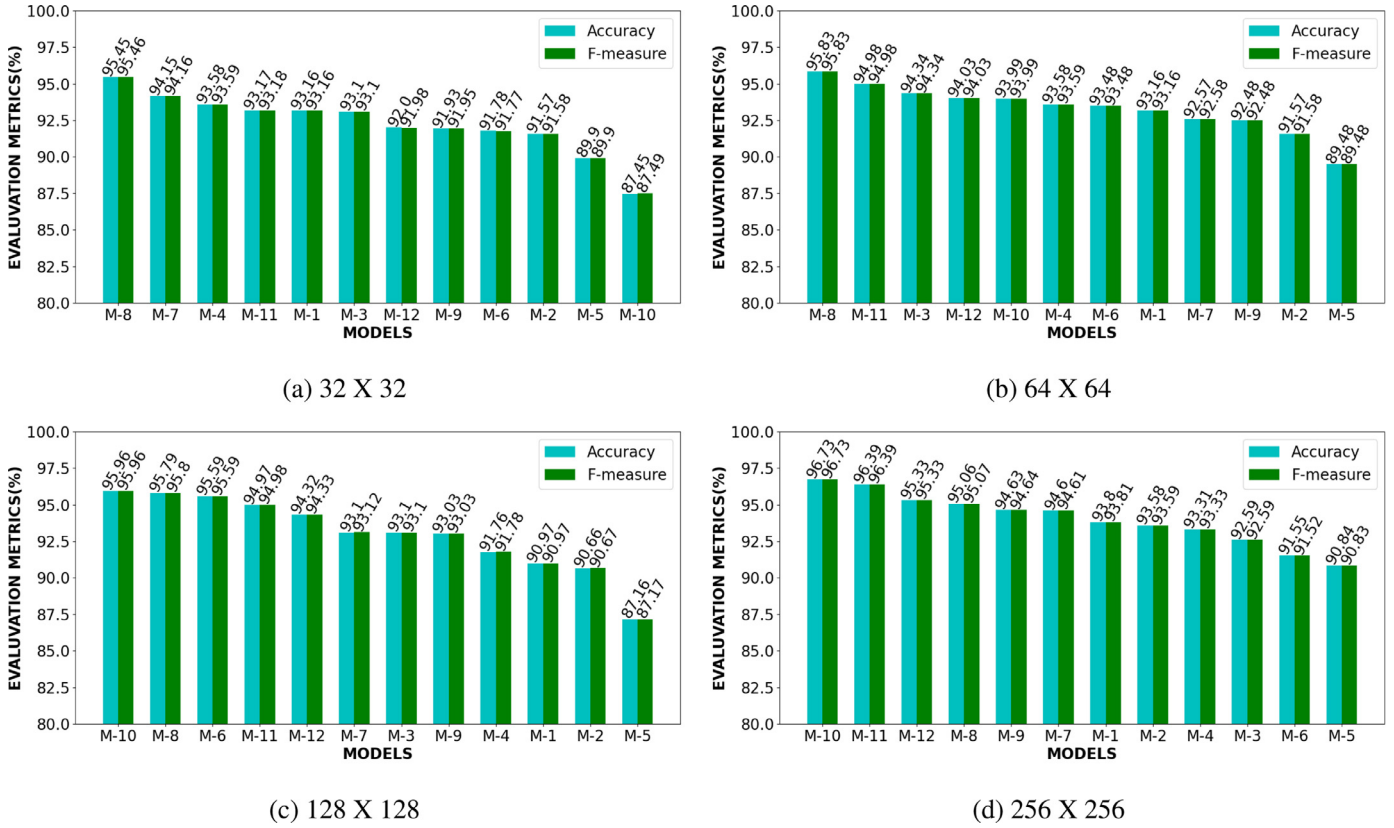
Performance of models generated using Gabor filter applied to grayscale images on Dataset-I and Dataset-II is depicted in Figs. 8 and 9. In 32 × 32, the F-measure obtained falls in the range from 87.49% to 95.46% for the Dataset-I and from 82.34% to 98.57% for the Dataset-II. F-measure on benign and BIG 2015 malware Dataset is 95.45% and 95.46% respectively and represents the highest accuracy. On benign and Malimg malware Dataset, both are 98.57%. The classification model M-8 (three pairs of convolution and pooling followed by a fully connected layer) exhibited improved predictions compared to all the other classifiers considered in the study on Dataset-I. On Dataset-II the model M-11 (VGG3 with dropout and batch normalization) performs as the best one.

Experiments were conducted on 64 × 64 grayscale where F-measure was found ranging from 89.48% to 95.83% on Dataset-I. For Dataset-II it ranges from 94.67% to 98.81%. The best classification model in the case of Dataset-I is M-8 (three groups of convolution and pooling followed by a fully connected layer) with 95.83% of accuracy and F-measure. Considering Dataset-II, the model M-10 (VGG3 with dropout) obtained 98.8% accuracy and F-measure respectively.

Additionally, we considered the image dimension of 128 × 128 and observed values of accuracy in between 87.16% and 95.96% on benign and BIG 2015 malware Dataset. The classification model M-10 (VGG3 with dropout) exhibited improved performances (95.96% accuracy and F-measure) compared to all other classifiers considered in the study on Dataset-I. In the case of benign and Malimg malware Dataset, the accuracy and F-measure vary from 92.0% to 99.01%. The model M-6 (two convolution layers followed by global pool and fully connected layer) produced 99.01% of accuracy and F-measure and shows the highest performance on Dataset-II.

Finally, performing experiments on the image dimension 256 × 256, we found that model M-10 (VGG3 with dropout) has the highest F-measure (96.73% on Dataset-I and 99.17% on Dataset-II). Considering Dataset-I, both accuracy and F-measure vary in the range 90.84% to 96.73% and for Dataset-II, from 90.72% to 99.17%.

The experiments were also performed using Gabor filter applied to color images of different dimensions (32 × 32, 64 × 64, 128 × 128 and 256 × 256) and identical trends were observed. The Table 5 represents some of the best outputs ob-



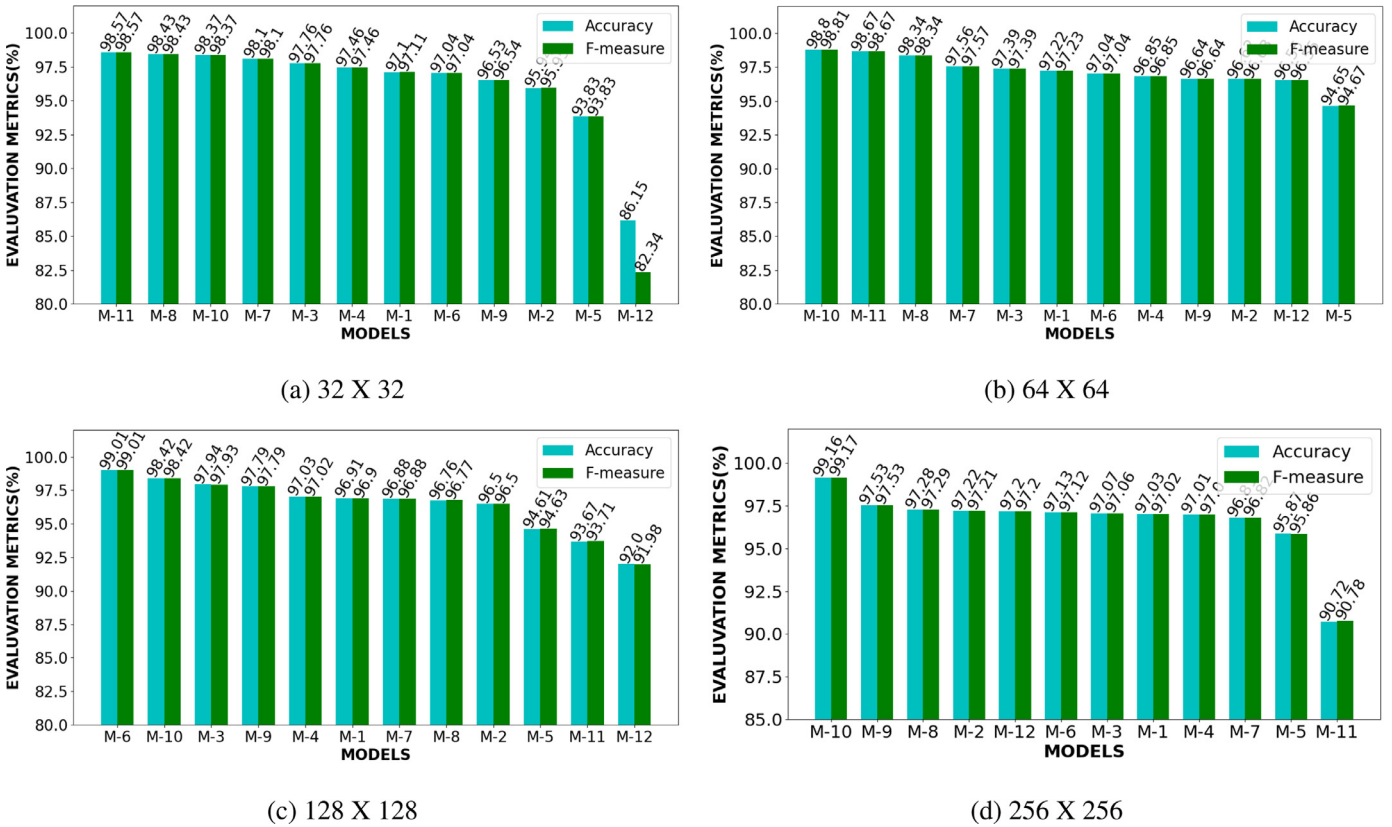
(a) 32 X 32

(b) 64 X 64

(c) 128 X 128

(d) 256 X 256

Fig. 8 – Performance of models generated using Gabor filter applied grayscale image on Dataset-I (Benign and BIG 2015 malware).



(a) 32 X 32

(b) 64 X 64

(c) 128 X 128

(d) 256 X 256

Fig. 9 – Performance of models generated using Gabor filter applied grayscale image on Dataset-II (Benign and Malimg malware).

Table 5 – The results of models generated using Gabor filter applied RGB images with dimension 256 x 256.

Dataset	Model	A (%)	F (%)
Dataset-I	M-11	95.98	95.98
	M-8	95.94	95.94
	M-10	95.80	95.80
	M-12	95.32	95.32
	M-9	94.67	94.67
Dataset-II	M-10	98.93	98.93
	M-11	98.65	98.65
	M-8	98.07	98.07
	M-12	97.46	97.45
	M-9	97.25	97.24

Table 6 – The results of models generated using Gabor filter applied Markov images with dimension 256 x 256.

Dataset	Model	A (%)	F (%)
Dataset-I	M-11	99.20	99.20
	M-10	98.59	98.59
	M-12	98.38	98.38
	M-9	98.29	98.30
Dataset-II	M-11	99.97	99.97
	M-10	99.70	99.70
	M-9	99.66	99.66
	M-12	99.61	99.61

tained. On Benign and BIG 2015 malware Dataset the accuracy and F-measure are in between 94.67% and 95.98%. Considering benign and Malimg malware Dataset, performance varies from 97.25% to 98.93%. The classification model M-11 (VGG3 with dropout and batch normalization) shows a high accuracy (95.98%) on Dataset-I. On Dataset-II model M-10 (VGG3 with dropout) has improved accuracy (98.93%).

We also experimented Gabor filter applied to Markov images and found that on Dataset-I, the model M-11 (VGG3 with dropout and batch normalization) exhibits accuracy and F-measure of 99.20% as discussed in Table 6. Similarly on Dataset-II, high accuracy and F-measure of 99.97% were produced by Model M-11 (VGG3 with dropout and batch normalization). The hyperparameters are epochs-30, learning rate-0.0001 and dropout rates 0.1, 0.2, 0.3. The Network topology was VGG3 along with three fully connected layers of 4096 neurons each. Here, in the case of Dataset-I, the accuracy and F-measure values vary in between 98.29% to 99.20% and from 99.61% to 99.97% on Dataset-II.

Comparing all the experiments, we observe that the classification model M-11 (VGG3 dropout with batch normalization), using Gabor filter applied Markov images, performs well on both the Datasets. This is because Markov image encompasses the probability values of paired consecutive sequences in an input file. We performed comprehensive analysis on various Gabor Markov images by adjusting parameter values of filters (refer Table 7). Finally, filter with kernel size = 3, standard deviation (σ) = 3, theta(θ) = 180, λ = 180, and aspect ratio = 0.5 was utilized to extract image textures, since collection of these images resulted in best performance on M11 model.

Table 7 – Parameter values used to generate Gabor Markov images.

Parameter Set	Kernel size	Standard deviation	Theta	Wavelength	Aspect ratio
P_1	3	3	180	180	0.5
P_2	3	4	120	120	0.4
P_3	3	5	90	90	0.3
P_4	3	6	60	60	0.2
P_5	3	7	30	30	0.1
P_6	5	3	45	45	0.5
P_7	3	2	20	20	0.01
P_8	3	1	10	10	0.001

Fig. 10 shows performance of Gabor Markov Images on different filter parameters (P_i) on M11 model, for image dimension 256 × 256 in Dataset-I and Dataset-II. Better outcomes are obtained with parameter values denoted by P_1 shown in Figs. 10, 11a and b.

4.4. *Malware classification*

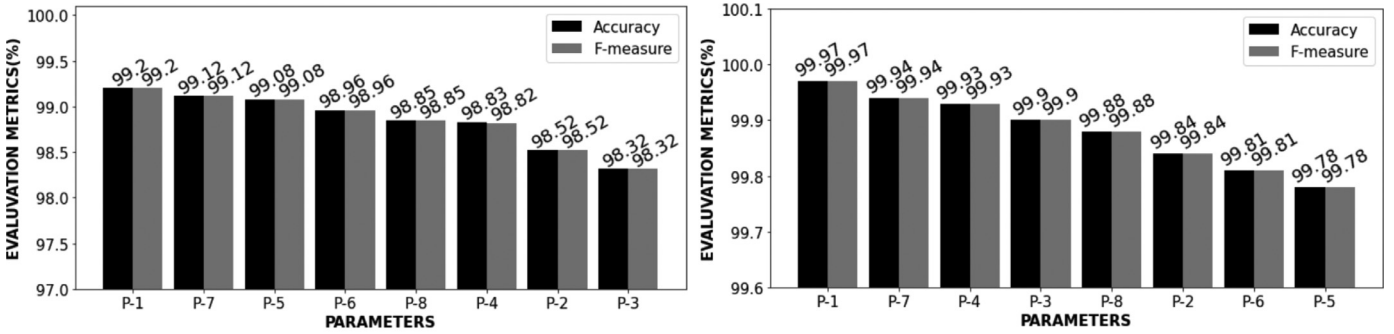
Malware classification is a key task of malware analysis, as demonstrated by the numerous results produced (Agarap and Pepito, 2018; Cui et al., 2018; Han et al., 2015; Karbab et al., 2018; Martinelli et al., 2017; McLaughlin et al., 2017; Nataraj et al., 2011; Pascanu et al., 2015; Rezende et al., 2018; Saxe and Berlin, 2015; Vinayakumar and Soman, 2018). The proposed method could be also used to determine the family a malware belongs to. We realized malware classification by plotting the entropy values corresponding to each segment (collection of bytes of a fixed sized block) of the binary file: these images are referred by us as entropy images. In particular, we divide the byte sequence of the executable sample into segments of 128 bytes. The elements of each segment (i.e., bytes) will vary in the range 0 to 255. Let us assume that an executable contains N segments/blocks represented as $S = \{s_1, s_2, s_3, s_4, \ldots, s_N\}$. We calculate the entropy of a segment s_j using Eq. (7)

$$H(s_j) = -\sum_{k=0}^{255} P(b_k j) \log P(b_k j) \tag{7}$$

where

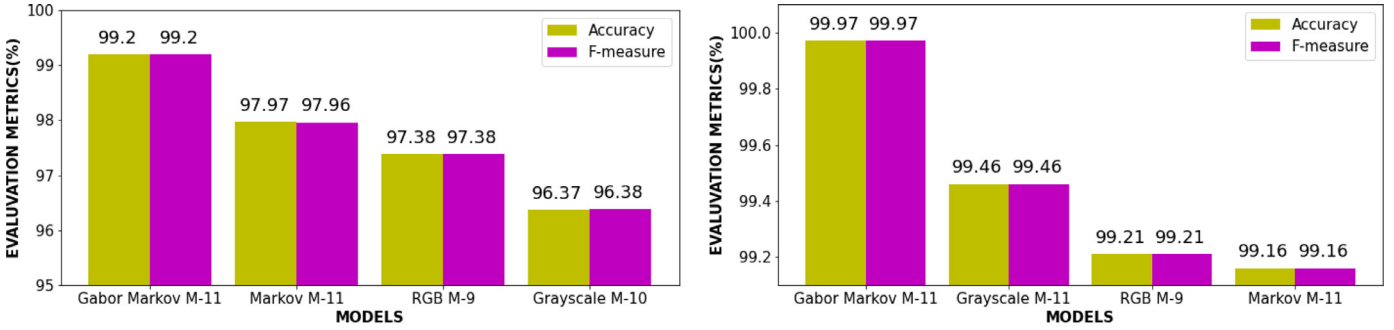
- $H(s_j)$ is the entropy of segment s_i.
- $P(b_k j)$ is the probability of byte b_k in segment s_j

Using the entropy values of segments, we generate the entropy images to ascertain whether the malware variants of a given family are visually identical. Figs. 12 and 13 exhibit the entropy images generated for different malware families on BIG 2015 and Malimg respectively. After analyzing the entropy images, it can be observed that variants belonging to the same family appear to be visually alike, while graphs of malware samples belonging to a different family are distinct. This visual similarity of graphs are clear indicators of the fact that the code reuse is largely employed by malware writers to create new instances. Additionally, it was seen that the different samples in a family exhibit similarity across various blocks/segments. Such blocks are spatially located in different



(a) Dataset-I (b) Dataset-II

Fig. 10 – Performance of Gabor Markov Images on different filter parameters on M11 model.



(a) Dataset-I (Benign and BIG 2015 malware) (b) Dataset-II (Benign and Malimg malware)

Fig. 11 – Performance comparison of best models on Dataset-I and Dataset-II.

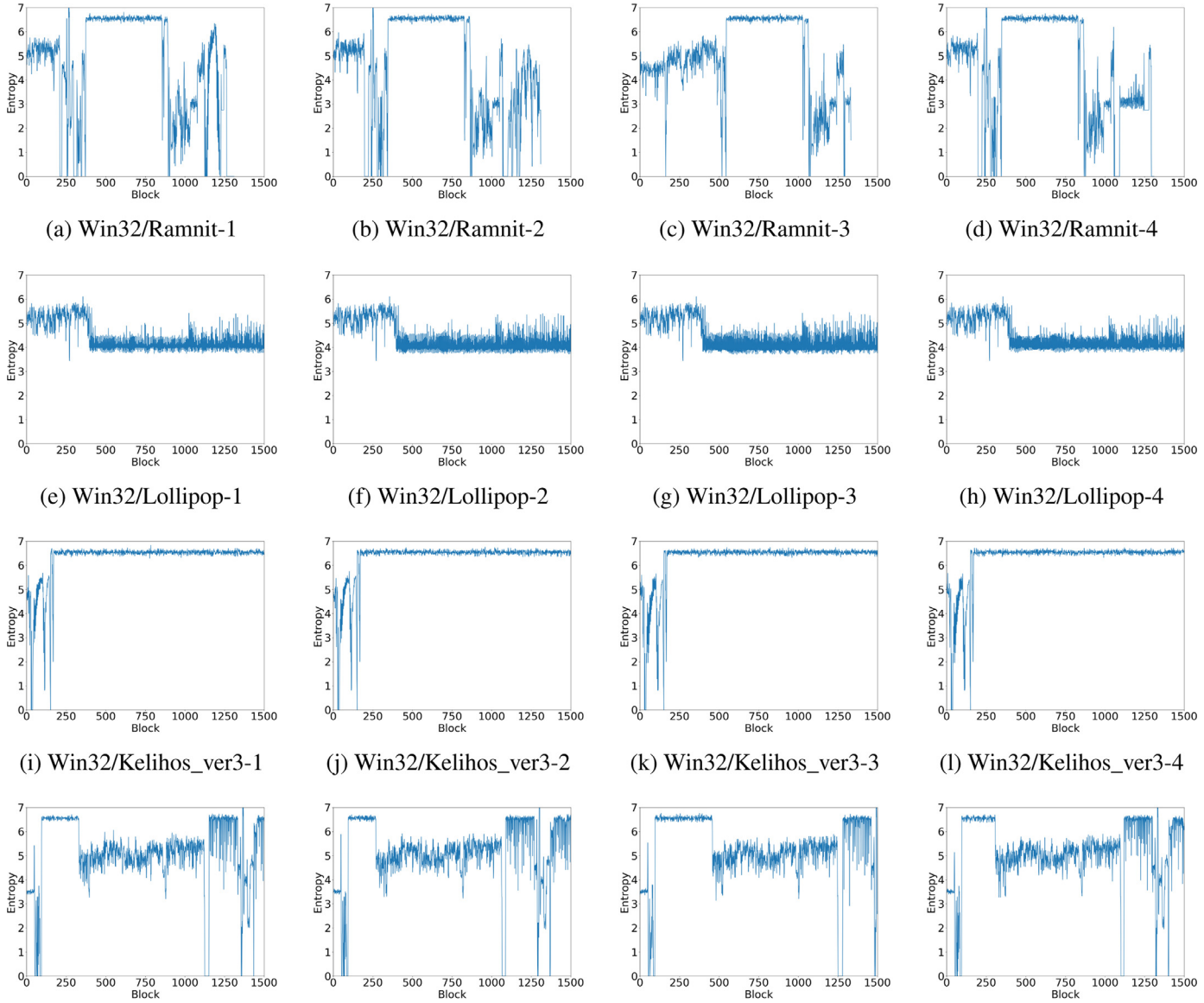


Fig. 12 – Entropy images of different malware families in BIG 2015 dataset.

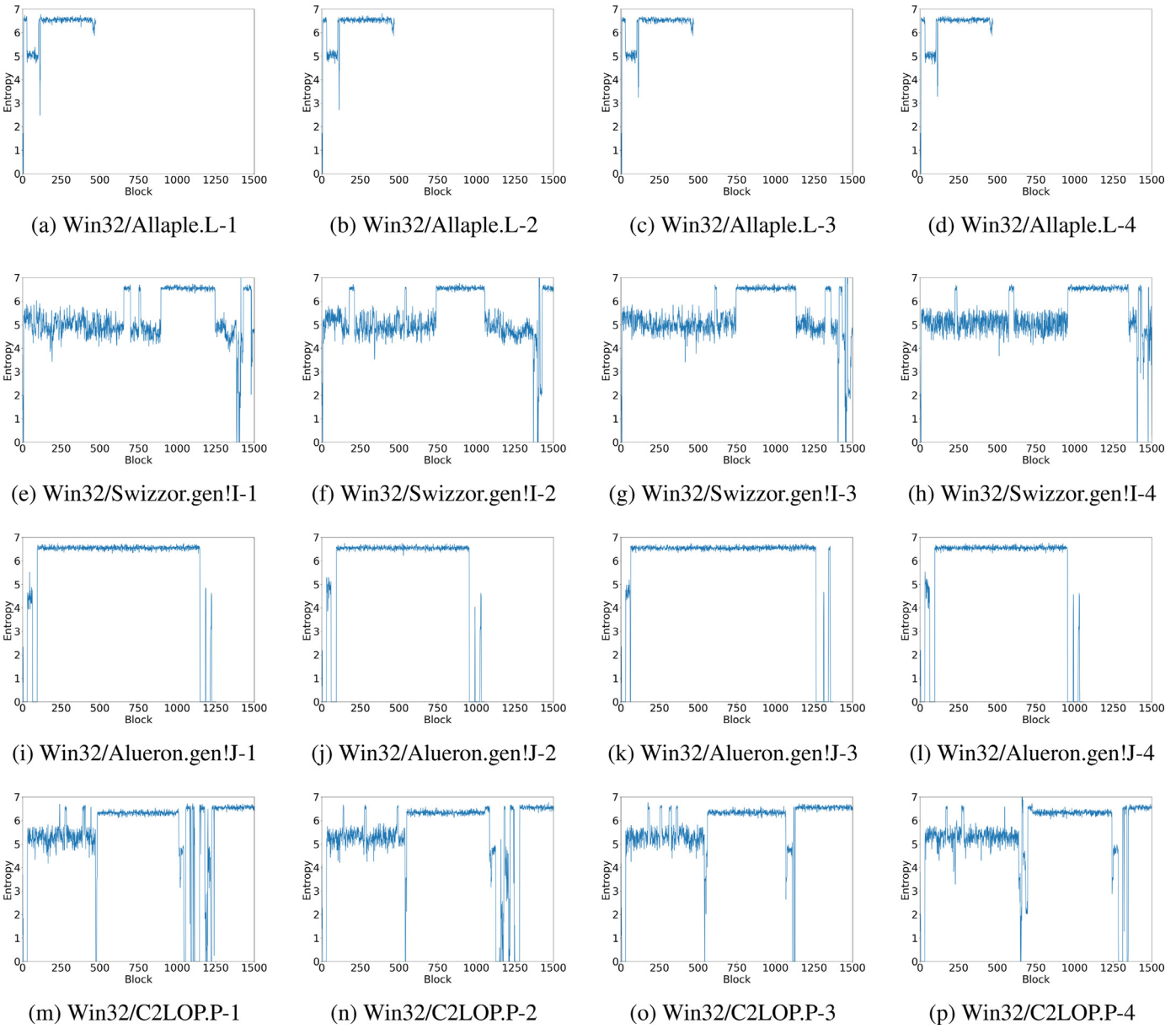


Fig. 13 – Entropy images of different malware families in Malimg dataset.

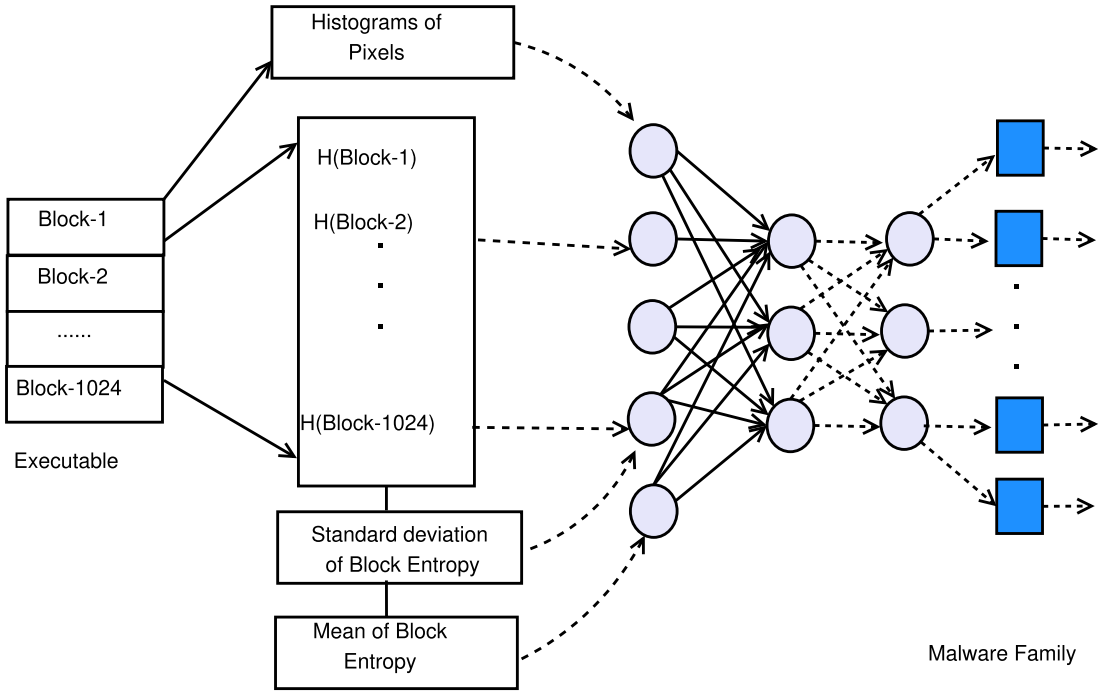


Fig. 14 – Malware family classification.

regions of a sample, indicating that malware samples mostly move the payload to confuse/defeat signature based detectors. Besides, the code movement is performed by incorporating conditional/unconditional branch instructions or utilizing function calls. Thus, entropy images gives an analyst clear clue of the segments that are closely similar in the majority of samples. Thus, knowledge of identical blocks can be used to develop block/segment signatures, for detecting new malware variants.

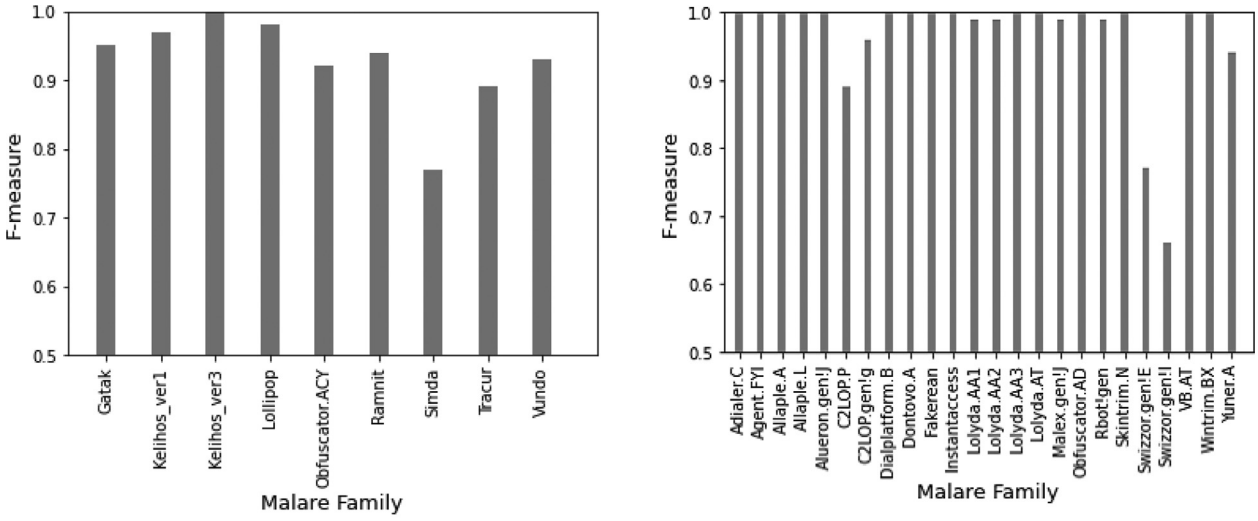
To empirically validate the results, malware binaries are divided into fixed blocks/segments. Initially, executables are grouped according to the file size. Let us consider that we have N files in the dataset, then we form different clusters of files, i.e., $C = \{c_1, c_2, \cdots, c_N\}$. All the files having identical size are grouped in c_i. Under certain cases, we pad the files to map it to the nearest cluster. Besides, we divided each executable to contain fixed-sized blocks irrespective of its size. By doing so, executables of large size contain bigger segments, conversely, small files contain shorter segments. For each segment (s_i), we determine entropy $H(s_i)$. Additionally, histograms of each pixel is computed.

Further, the mean and standard deviation of block entropy is estimated. Thus, we arrive at a feature space of 1282 dimensions (1024-block entropy values, histograms of the pixel having 256 dimensions and two statistical attributes, i.e., mean and standard deviation). Using 1282 attributes, a seven-layer deep neural network is trained with an output layer consisting of neuron equivalent to the number of families. Finally, unseen samples are assigned to their respective family. The process is shown in Fig. 14. After evaluating different configurations of DNN structure along-with hyper-parameters, we found that the seven-layers DNN produced a weighted average of 95.94% for the accuracy and 95.92% as the F-measure with BIG 2015 dataset, accuracy of 97.68% and F-measure of 97.11% for the Malimg dataset.

From Fig. 15a, we observe the F-measure exceeding 91% except for the families Simda and Tracur. As the dataset is highly imbalanced, samples with fewer malware variants in a family were observed to be misclassified. Notably, the test set of Simda consisted of 13 instances, out of which 2 instances were wrongly labelled as Tracur and 1 instance was misclassified as Ramnit. Hence, the precision and recall values for the Simda family is 77%. The precision obtained for Tracur malware family is 90%.

In the case of Malimg dataset (refer to Fig. 15b), we observed that except for samples of Autorun.K, C2LOP.P, and Swizzor families, all the malware variants were precisely classified into their respective classes with an F-measure exceeding 94 %. Specific to the aforesaid families, we noticed that the number of samples were in range of 106 to 146, out of which the 70% of samples are presented to classifier during training. Autorun.K was wrongly labelled as Yuner.A due to the increased similarity between the samples. This imbalance nature in the dataset hinders the performance of a classifier, further increases the complexity of extracting patterns for appropriately discriminating binaries of different malware families.

We extended our experiments for improving the outcome of the malware classification. Motivated by the visual similarity of variants in a family (refer Figs. 12 and 13), the malware classification problem was addressed by considering images of entropy as input to the deep learning network. By analyzing the outcomes of all the classification models, we finally decided to investigate the deep learning model that gave the best results in the majority of the experiments we conducted previously, hence we choose tuned M11 model (learn-



(a) BIG 2015 dataset (b) Malimg dataset

Fig. 15 – Malware family classification.

Fig. 16 – The confusion matrix of malware family classification on BIG 2015 dataset.

ing rate = 0.001, epochs = 30 and batch-size = 32). To be precise, we trained the classification module with the figures (entropy images) generated with Matplotlib (a plotting library in Python). On BIG 2015 and Malimg dataset, we obtained an F-measure of 98.25% and 98.38% (shown in Table 8), which is higher than the output obtained with previous experiments performed with seven layer DNN (F-measure for Dataset-I is 97.0% and for Dataset-II is 95.0%). Figs. 16 and 17 illustrate the confusion matrix of Model M-11 on BIG 2015 and Malimg dataset. From Fig. 17, we observe that 42.5% of the samples used for the prediction from Swizzor.gen!I family was misclassified as C2LOP.gen!g, and 15% into Swizzor.gen!E family. Further, we computed the number of blocks in misclassified samples where similarity is higher than 60% (refer to Fig. 18).

Table 8 – Results of malware family classification using model M-11.

Dataset	*A (%)*	*F (%)*
BIG 2015	98.25	98.25
Malimg	98.46	98.38

For samples misclassified as C2LOP.gen!g, 51 301 blocks were similar (refer to Fig. 18a). Similarly, for samples misclassified as Swizzor.gen!E 187 299 similar blocks were observed (refer to Fig. 18b).

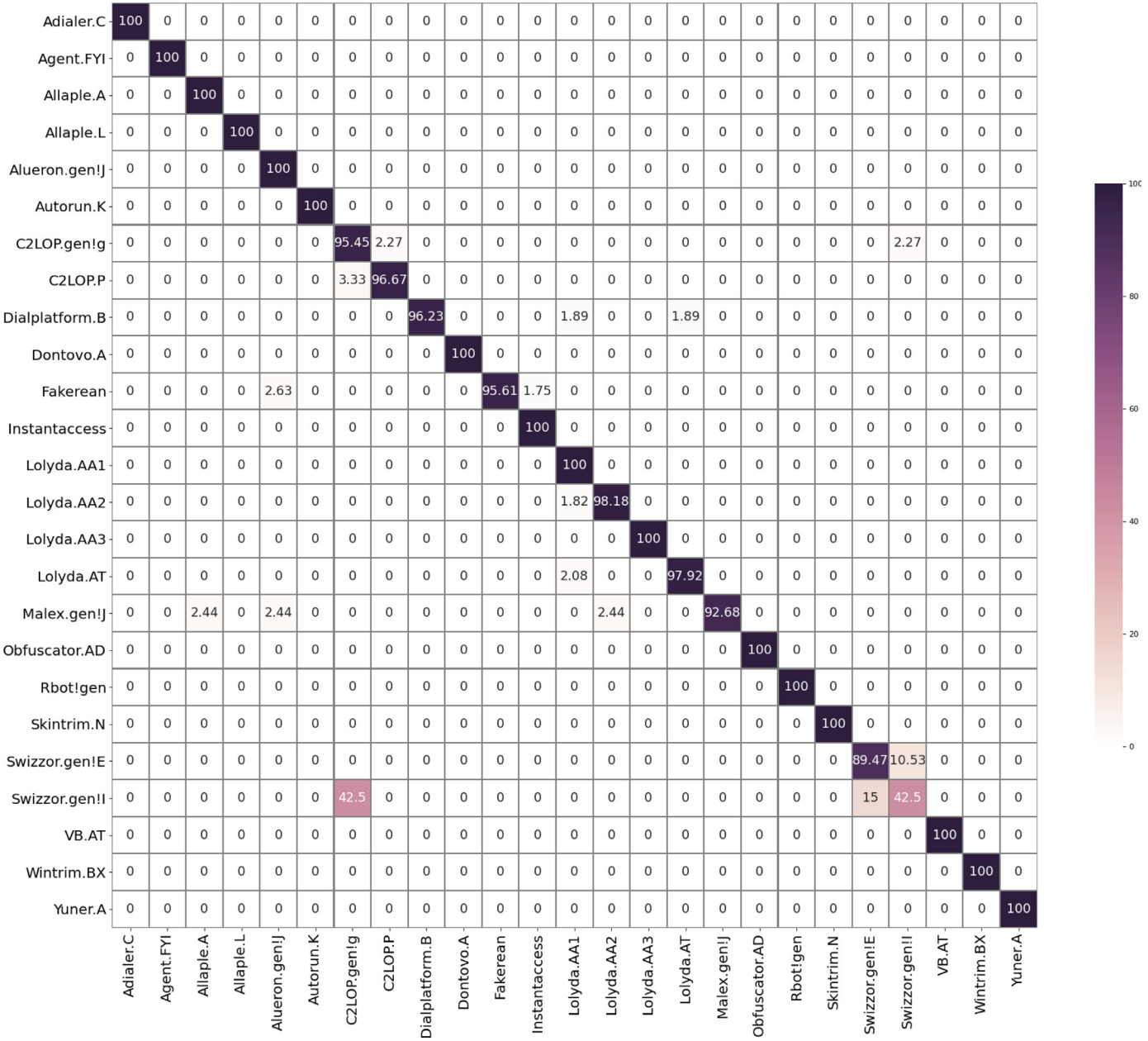
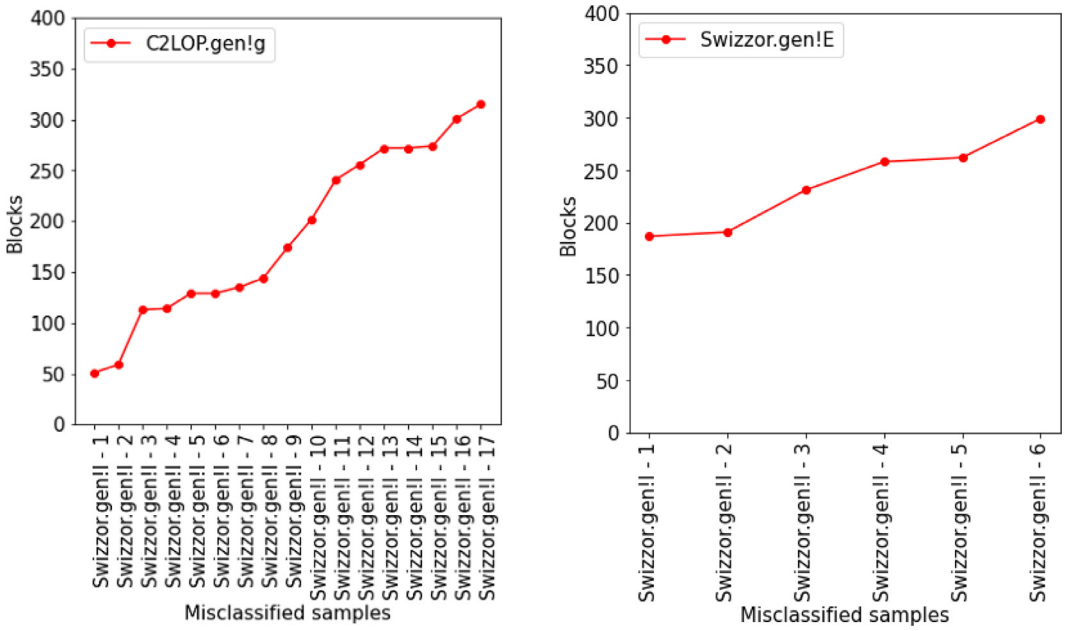


Fig. 17 – The confusion matrix of malware family classification on Malimg dataset.



(a) (b)

Fig. 18 – Block similarity of Swizzor.gen!I samples misclassified as C2LOP.gen!g and Swizzor.gen!E families.

5. Discussion

From the experimental trials, it emerged that the models trained on the image dimension 256×256 perform better than the ones trained on the image dimensions like 32×32, 64×64, 128×128 for both grayscale and RGB image. However, models with smaller image dimensions are indeed able to capture the global features of an image rather than the fine-grained features. Since malware is often written by inserting malicious code into benign samples, malware and benign samples are visually similar, and fine-grained features are required for improving the classification. Images with higher image dimensions are able to accommodate more pixels and thereby more fine-grained features can be captured by CNN.

It can also be noted that among all the models trained using grayscale and Markov images on both datasets, models M-10 (VGG3 with dropout) and M-11 (VGG3 with dropout and batch normalization) gave better results. The effect of regularization techniques like dropout (Srivastava et al., 2014) and batch normalization is evident in this outcome.

From all the experiments presented in Section 4 on diverse images dimensions, we deduce that Markov images produce the best results. Markov image represents the semantic structure of the binary sequence of the executable file. Images like grayscale and RGB contain raw and unstructured information which makes it hard for CNN to extract the local features. Even though grayscale and RGB images produce good results, the layers in the neural network cannot filter relevant patterns compared to the Markov images. As we chose two bytes to create a pixel using colormap, fraction of original information is lost. Since the dimensions of pictures generated are different due to differences in file size, down-sampling the original image to image dimensions 32×32, 64×64, 128×128, and 256×256 result in loss of information. For larger size image, the features get lost due to the inability to represent the features in a limited number of pixels. In the case of smaller size files, down-sampling causes the deformation of patterns and textures across the image.

From all experiments, it is observed that the images (both gray scale and RGB images) yielded better results by carefully tuning deep neural networks. Improved outcomes are observed if the CNN architectures were presented with Gabor images (i.e., image obtained on the application of Gabor filter). As a malware variant inherits code from the base sample, and in particular code reuse is predominant (Calleja et al., 2019; Lee, 2015), in variants of a specific malware family, the texture analysis revealed the hidden patterns and this improves consequently the prediction.

Since textures and patterns obtained depend on the chosen width, there is some arbitrariness involved in selecting the width during the image generation. Therefore, a rational approach which portrays the semantics of the sequence was required in generating the image, these limitations are resolved by transforming binaries as malware Markov images.

We extended our research to malware family classification. Initially, we plotted entropy images of all the samples. We observe that malware variants belonging to the same family are visually similar, while those belonging to different families exhibit clear differences. Additionally, code reuse is predominant in the generation of new instances within the family. Furthermore, we trained seven-layers DNN (using histogram of pixels, block entropies, standard deviation of block entropy and mean of block entropy). However, samples of certain

Table 9 – Comparison of our methodology with other state-of-the-art malware detection models.

Malware Detection Models	Malware Prediction (P)/ Classification (C)	Description	A(%)
	State-of-the-art		
Malware Images: Visualization and Automatic Classification (2011) (Nataraj et al., 2011)	C	Malimg malware binaries (9339 samples) are read as a vector of 8 bit unsined integers. GIST descriptors with 8 orientations and 4 scales were computed on the images.Finally, k-nearest neighbors with Euclidean distance was employed for predicting samples in the test set.	98.08
Malware Analysis and Classification using Artificial Neural Networks (2015) (Makandar and Patrot, 2015)	C	A total of 3131 Mahenhur malware samples are represented as 2D gray scale image. Gabor wavelet and GIST descriptors were used to derive 320 features and classification models were developed using ANN.	96.35
Detection of Malicious Code Variants Based on Deep Learning (2018) (Cui et al., 2018)	C	Malware binary samples from Malimg dataset were splitted into a number of substrings of 8 bits length and were further represented into two-dimensional (2-D) grayscale images. A Convolutional neural network (CNN) was employed to classify malware images into its respective families.	94.5
Deep learning at the shallow end: Malware Classification for non-domain experts (2018) (Le et al., 2018)	C	Binary representation of BIG 2015 (10860 samples) was utilized, and experiments were performed using six deep learning configurations, each of which was a combination of one of three deep learning architectures (CNN, CNN-UniLSTM, CNN-BiLSTM).	98.2
Malicious Software Classification using VGG16 Deep Neural Network's Bottleneck Features (2018) (Rezende et al., 2018)	C	The byteplot representation of 10136 malware executables from VirusSign was used for identification of visual patterns in the samples. The byteplot image with one channel was subsequently converted into 224 × 224 resolution RGB image. Finally, the mean RGB values of pixels were used as features in the proposed DNN model.	92.97
Towards Building an Intelligent Anti-Malware System: A Deep Learning Approach using Support Vector Machine (SVM) for Malware Classification (Agarap and Pepito, 2018)	C	Gray scale images of Malimg dataset is used, and classification models were developed using CNN- SVM (2-layer), GRU-SVM (5-layer) and MLP-SVM (3-layer).	84.92
Windows Malware Detector using Convolutional Neural Network based on Visualization Images (2019) (Shiva Darshan and Jaidhar, 2019)	P	A collection of 3282 Benign samples and 4151 Malware samples were used to CNN-based detector. API calls and their corresponding category were utilized as input features. The Image Generator identifies the occurrence of a N-gram of attributes to construct an image.	97.968
	Our Methodology		
Grayscale Image Approach	P	Benign-12971 samples Malimg malware-9339 samples	99.46
		Benign-12971 samples BIG 2015 malware-10860 samples	96.37
RGB Image Approach	P	Benign-12971 samples Malimg malware-9339 samples	99.21

(*continued on next page*)

Table 9 (*continued*)

Malware Detection Models	Malware Prediction (P)/ Classification (C)	Description	A(%)
		Benign-12971 samples BIG 2015 malware-10860 samples	97.38
Markov Image Approach	P	Benign-12971 samples Malimg malware-9339 samples	99.16
		Benign-12971 samples	
BIG 2015 malware-10860 samples	97.97		
Gabor filter applied Image Approach	**P**	**Benign-12971 samples Malimg malware-9339 samples**	**99.97**
		Benign-12971 samples BIG 2015 malware-10860 samples	**99.20**
Entropy Image Approach	C	Malimg malware-9339 samples	98.46
		BIG 2015 malware-10860 samples	98.25

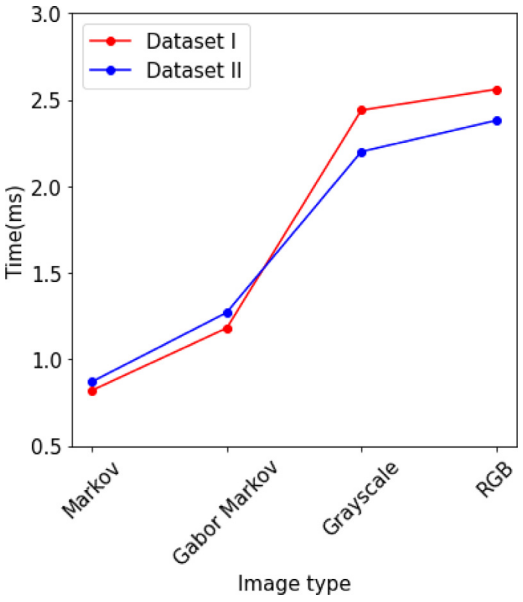


Fig. 19 – Plot showing the execution time consumed by the best models in four different approaches on Dataset-I and Dataset-II.

families were misclassified. In order to improve the performance measures, we trained the model M-11 (VGG3 dropout with batch normalization) with entropy images on both the datasets, and thus we obtained a higher F-measure and a higher accuracy. The convolutional layers in VGG3 extracts distinguishing features from the input entropy images, which improves malware family classification.

Fig. 19 shows time consumed by the best models in predicting individual samples for various image representations, i.e., Grayscale, RGB, Markov and Gabor Markov on both datasets. We can observe that the approach using Markov has a faster detection speed with 0.82 ms for Dataset I and 0.87 ms for Dataset II compared to other models.

6. Comparative analysis

In this paper we propose a model to detect and classify malware by using visualization techinques. To highlight how our model outperforms the current state-of-the-art systems, we realized a comparative analysis with similar existing malware detection models. Form Table 9 it can be observed that the accuracy values of the state-of-the-art systems fall in the range from 84.92% to 98.08%. We obtained an accuracy of 99.97%. Moreover, for malware identification, the prior work (Shiva Darshan and Jaidhar, 2019) demonstrates an accuracy of 97.97% with fewer samples.

7. Conclusion and future scope

In this paper, we perform comprehensive analysis of visual images derived from executables of different dimensions, for appropriately identifying malware. This is achieved by evaluating the performance of various fine tuned CNN models. Our study showed the best results with coloured visual images of size 256 × 256. The experiments performed on two benchmark data sets resulted in a F-measure of 97.38% (for Dataset-I) and 99.21% (for Dataset-II) respectively. In order to further improve the results, we represented executables as Markov images, where each pixel correspond to the probability of the occurrence of a pair of bytes in a file. The classifier trained with Markov images exhibited better accuracies and F-measure comparing coloured and gray scale images. For Dataset-I and Dataset-II for Markov images of dimension 256×256 the best F-measures obtained were 97.96% and 99.16% respectively. Additionally, extensive analysis for improving the results were conducted by presenting diverse configurations of images to the CNN architectures. To carry out the aforementioned analysis, we additionally extracted textures from images by the application of Gabor filter (thus referred such images as Gabor images). Our study demonstrates that Gabor images derived from Dataset-I produced an F-measure of 99.20%, besides for Dataset-II, accuracy and F-measures of 99.97% respectively. Finally, we segmented an executable into blocks, computed

block entropy, and represented the entropy of blocks in the form of images. We found that, malware variants within a specific family showed similarity in entropy images, demonstrating repetition of block of pixels/frequent patterns. The best CNN models in majority of the cases are M-9 (VGG3 baseline), M-10 (VGG3 with dropout), M-11 (VGG3 with dropout and batch normalization) and M-12 (ResNet-50).

In future, we would like to extend our study by experimenting on a much larger set. It is also intended to investigate the performance of multimodal image classifier, where the base classifier will be trained on the combination of images comprising of textures, entropy images, Markov images and localized images patterns. Subsequently, the patterns learned by the hidden layers would be used to train meta-DNN classifier. Additionally, we would like to study the movement of byte patterns using optical flow to understand the locomotion of code segments across variants within a malware family.

Funding

None.

Declaration of Competing Interest

None.

CRediT authorship contribution statement

Anson Pinhero: Writing - original draft. **Anupama M L:** Writing - original draft. **Vinod P:** Conceptualization, Methodology, Validation, Writing - review & editing. **C.A. Visaggio:** Supervision, Methodology, Validation, Writing - review & editing. **Aneesh N:** Software, Writing - original draft. **Abhijith S:** Software, Writing - original draft. **AnanthaKrishnan S:** Software, Writing - original draft.

REFERENCES

Abbas Alipour A, Ansari E. An advanced profile hidden Markov model for malware detection; 2019.

Agarap AF, Pepito FJH. Towards building an intelligent anti-malware system: a deep learning approach using support vector machine (SVM) for malware classification. CoRR 2018 abs/1801.00318.

Akarsh S, Simran K, Poornachandran P, Menon VK, Soman KP. Deep learning framework and visualization for malware classification. In: 2019 5th International Conference on Advanced Computing Communication Systems (ICACCS); 2019. p. 1059–63.

Arefkhani M, Soryani M. Malware clustering using image processing hashes. In: 2015 9th Iranian Conference on Machine Vision and Image Processing (MVIP); 2015. p. 214–18.

Anubis, http://anubis.cs.ucsb.edu/.

Bläsing T, Batyuk L, Schmidt A, Camtepe SA, Albayrak S. An android application sandbox system for suspicious software detection. In: 2010 5th International Conference on Malicious and Unwanted Software; 2010. p. 55–62.

Burguera I, Zurutuza U, Nadjm-Tehrani S. Crowdroid: behavior-based malware detection system for android; 2011. p. 15–26.

Calleja A, Tapiador J, Caballero J. The malsource dataset: quantifying complexity and code reuse in malware development. IEEE Trans. Inf. ForensicsSecur. 2019;14(12):3175–90.

Chan P, Yiu S. DroidChecker: analyzing android applications for capability leak. WiSec'12 - Proceedings of the 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks, 2012.

Chuang H, Wang S. Machine learning based hybrid behavior models for android malware analysis. In: 2015 IEEE International Conference on Software Quality, Reliability and Security; 2015. p. 201–6.

Cui Z, Xue F, Cai X, Cao Y, Wang G, Chen J. Detection of malicious code variants based on deep learning. IEEE Trans. Ind. Inf. 2018;14(7):3187–96.

David B, Filiol E, Gallienne K. Structural analysis of binary executable headers for malware detection optimization. J. Comput. Virol. Hacking Tech. 2016;13. doi:10.1007/s11416-016-0274-2.

David OE, Netanyahu NS. DeepSign: deep learning for automatic malware signature generation and classification. In: 2015 International Joint Conference on Neural Networks (IJCNN); 2015. p. 1–8.

Driverpack Solution; 2020. (accessed February 18) https://drp.su/en

Goodall J, Radwan H, Halseth L. Visual analysis of code security; 2010. doi:10.1145/1850795.1850800.

Grigorescu SE, Petkov N, Kruizinga P. Comparison of texture features based on Gabor filters. IEEE Trans. Image Process. 2002;11(10):1160–7.

Han EGIKS, JH Lim. Malware analysis method using visualization of binary files. In: Proceedings of the 2013 Research in Adaptive and Convergent Systems; 2013. p. 317–21.

Han KS, Lim JH, Kang K, Im EG. Malware analysis using visualized images and entropy graphs. Int. J. Inf. Secur.Springer 2015;14(1):1–14.

He K, Kim D. Malware detection with malware images using deep learning techniques. In: 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE); 2019. p. 95–102.

He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2016. p. 770–8.

Hu W, Tan Y. Black-box attacks against RNN based malware detection algorithms; 2017.

Huang C-Y, Tsai Y-T, Hsu C-H. Performance evaluation on permission-based detection for Android malware, vol. 21; 2013. p. 111–20.

IDA pro, Hex Rays. https://www.hex-rays.com/products/ida.

Jang J-w, Yun J, Mohaisen A, Woo J, Kim HK. Detecting and classifying method based on similarity matching of Android malware behavior with profile. SpringerPlus 2016;5:1–23. doi:10.1186/s40064-016-1861-x.

Jang S, Li S, Sung Y. Fasttext-based local feature visualization algorithm for merged image-based malware classification framework for cyber security and cyber defense. Mathematics 2020;8:460. doi:10.3390/math8030460.

Jun-ling W, Shuo-hao W. Malicious classification based on deep learning and visualization. In: 2019 2nd International Conference on Artificial Intelligence and Big Data (ICAIBD); 2019. p. 223–8.

Kalash M, Rochan M, Mohammed N, Bruce NDB, Wang Y, Iqbal F. Malware classification with deep convolutional neural networks; 2018. p. 1–5.

Kancherla K, Mukkamala S. Image visualization based malware detection. In: 2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS); 2013. p. 40–4.
Kang B, Kim T, Kwon H, Choi Y-s, Im EG. Malware classification method via binary content comparison. In: Proceeding of the 2012 ACM Research in Applied Computation Symposium, RACS 2012; 2012. p. 316–21.
Karbab E, Debbabi M, Derhab A, Mouheb D. MalDozer: automatic framework for Android malware detection using deep learning. Digit. Invest. 2018;24:S48–59.
Kim T, Kang B, Rho M, Sezer S, Im EG. A multimodal deep learning method for android malware detection using various features. IEEE Trans. Inf. ForensicsSecur. 2019;14(3):773–88.
Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ, editors. In: Advances in Neural Information Processing Systems 25. Curran Associates, Inc.; 2012. p. 1097–105.
Kumar R, Khan RU. Opcode and gray scale techniques for classification of malware binaries; 2018.
Le Q. Malware identification using visualization images and deep learning. Comput. Secur. 2018;77:871–85.
Le Q, Boydell O, Mac Namee B, Scanlon M. Deep learning at the shallow end: malware classification for non-domain experts. Digit. Invest. 2018. doi:10.1016/j.diin.2018.04.024.
Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc. IEEE 1998;86(11):2278–324.
Lee T. Automatic malware mutant detection and group classification based on the n-gram and clustering coefficient. J. Supercomput. 2015. doi:10.1007/s11227-015-1594-6.
Li J, Sun L, Yan Q, Li Z, Srisa-an W, Ye H. Significant permission identification for machine-learning-based Android malware detection. IEEE Trans. Ind. Inf. 2018;14(7):3216–25.
Lin C-H, Pao H-KK, Liao J-W. Efficient dynamic malware analysis using virtual time control mechanics. Comput. Secur. 2018;73:359–73.
Liu Y, Zhang Y, Li H, Chen X. A hybrid malware detecting scheme for mobile Android applications. In: 2016 IEEE International Conference on Consumer Electronics (ICCE); 2016. p. 155–6.
Makandar A, Patrot A. Malware analysis and classification using artificial neural network; 2015. p. 1–6.
Martinelli F, Marulli F, Mercaldo F. Evaluating convolutional neural network for effective mobile malware detection. Procedia Comput. Sci. 2017;112:2372–81.
McLaughlin N, Doupé A, Ahn G, Martinez-del Rincon J, Kang B, Yerima S, Miller P, Sezer S, Safaei Y, Trickel E, Zhao Z. Deep Android malware detection; 2017. p. 301–8.
Mitsuhashi R, Shinagawa T. High-accuracy malware classification with a malware-optimized deep learning model; 2020.
Narayanan BN, Davuluru VSP. Ensemble malware classification system using deep neural networks. Electronics 2020;9:721. doi:10.3390/electronics9050721.
Nataraj L, Karthikeyan S, Jacob G, Manjunath B. Malware images: visualization and automatic classification; 2011.
Nataraja L, Jacobb G, Manjunatha BS. Detecting packed executable based on raw binary data; 2010.
Olshausen BA, Field DJ. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. Nature 1996;381(6583):607–9.
Pascanu R, Stokes JW, Sanossian H, Marinescu M, Thomas A. Malware classification with recurrent networks. In: Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP); 2015. p. 1916–20.
Patel K, Buddhadev B, vol. 536; 2015. doi:10.1007/978-3-319-22915-7_41.
Portable freeware collection; 2020. https://www.portablefreeware.com/(accessed February 18, 2020)
Pendlebury F, Pierazzi F, Jordaney R, Kinder J, Cavallaro L. TESSERACT: eliminating experimental bias in malware classification across space and time. In: Heninger N, Traynor P, editors. In: 28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019. USENIX Association; 2019. p. 729–46.
Qiao Y, Jiang Q, Jiang Z, Gu L. A multi-channel visualization method for malware classification based on deep learning. In: 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE); 2019. p. 757–62.
Qiao Y, Yang Y, He J, Tang C, Liu Z. CBM: free, automatic malware analysis framework using API call sequences. Knowl. Eng. Manage. 2014;214:225–36. doi:10.1007/978-3-642-37832-4_21.
Quist DA, Liebrock LM. Visualizing compiled executables for malware analysis. In: 2009 6th International Workshop on Visualization for Cyber Security; 2009. p. 27–32.
Raff E, Barker J, Sylvester J, Brandon R, Catanzaro B, Nicholas C. Malware detection by eating a whole exe; 2017.
Ren Z, Chen G, Lu W. Malware visualization methods based on deep convolution neural networks. Multimed. Tools Appl. 2019;79:1–19. doi:10.1007/s11042-019-08310-9.
Rezende E, Ruppert G, Carvalho T, Ramos F, De Geus P. Malicious software classification using transfer learning of resnet-50 deep neural network; 2017. doi:10.1109/ICMLA.2017.00-19.
Rezende E, Ruppert G, Carvalho T, Theophilo A, Ramos F, de Geus P. Malicious software classification using VGG16 deep neural network's bottleneck features. Inf. Technol.-New Gener. 2018:51–9.
Ronen R, Radu M, Feuerstein C, Yom-Tov E, Ahmadi M. Microsoft malware classification challenge; 2018 1802.10135.
Salehi M, Amini M. Android malware detection using markov chain model of application behaviors in requesting system services; 2017.
Saxe J, Berlin K. Deep neural network based malware detection using two dimensional binary program features. In: 2015 10th International Conference on Malicious and Unwanted Software (MALWARE); 2015. p. 11–20.
Schultz MG, Eskin E, Zadok F, Stolfo SJ. Data mining methods for detection of new malicious executables. In: Proceedings 2001 IEEE Symposium on Security and Privacy. S P 2001; 2001. p. 38–49.
Shabtai A, Kanonov U, Elovici Y, Glezer C, Weiss Y. Andromaly: a behavioral malware detection framework for Android devices. J. Intell. Inf. Syst. 2010;38:161–90.
Shijo P, Salim A. Integrated static and dynamic analysis for malware detection. Procedia Comput. Sci. 2015;46:804–11. doi:10.1016/j.procs.2015.02.149.
Shiva Darshan SL, Jaidhar CD. Windows malware detector using convolutional neural network based on visualization images. IEEE Trans. Emerg. Top. Comput. 2019:1.
Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. International Conference on Learning Representations, 2015.
Softonic; 2020. https://en.softonic.com/windows(accessed March 2, 2020)
Sourceforge; 2020. https://sourceforge.net(accessed March 2, 2020)
StatCounter Global Stats. Desktop operating system market share worldwide - May; 2020. https://gs.statcounter.com/os-market-share/desktop/worldwide
Symantec. Internet security threat report; 2019. https://www.symantec.com

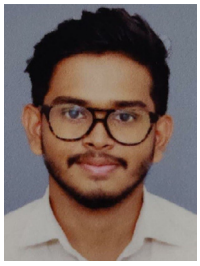
Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 2014;15(1):1929–58.
Sun G, Qian Q. Deep learning and visualization for identifying malware families. IEEE Trans. Dependable Secure Comput. 2018:1.
Taheri R, Ghahramani M, Javidan R, Shojafar M, Pooranian Z, Conti M. Similarity-based Android malware detection using hamming distance of static binary features; 2019.
Tajoddin A, Jalili S. HM3alD: polymorphic malware detection using program behavior-aware hidden markov model. Appl. Sci. 2018;8:1044. doi:10.3390/app8071044.
Tobiyama S, Yamaguchi Y, Shimada H, Ikuse T, Yagi T. Malware detection with deep neural network using process behavior, vol. 2; 2016. p. 577–82.
Turner M. Texture discrimination by Gabor functions. Biol. Cybern. 1986;55:71–82. doi:10.1007/BF00341922.
Venkatraman S, Alazab M, R V. A hybrid deep learning image-based analysis for effective malware detection. J. Inf. Secur. Appl. 2019;47:377–89. doi:10.1016/j.jisa.2019.06.006.
Vinayakumar R, Soman K. DeepMalNet: evaluating shallow and deep networks for static pe malware detection. ICT Express 2018;4:255–8.
VirusTotal; 2020. https://www.virustotal.com(accessed February 18, 2020)
Wang X, Yang Y, Zeng Y, Tang C, Shi J, Xu K. A novel hybrid mobile malware detection system integrating anomaly detection with misuse detection; 2015. p. 15–22.
Wu W-C, Hung S-H. In: RACS '14. DroidDolphin: a dynamic Android malware detection framework using big data and machine learning; 2014.
Xiao X, Wang Z, Li Q, Xia S, Jiang Y. Back-propagation neural network on markov chains from system call sequences: a new approach for detecting Android malware with system call sequences. IET Inf. Secur. 2017;11(1):8–15.
Yakura H, Shinozaki S, Nishimura R, Oyama Y, Sakuma J. Malware analysis of imaged binary samples by convolutional neural network with attention mechanism. Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, 2018.
Yan J, Qi Y, Rao Q. Detecting malware with an ensemble method based on deep neural network. Secur. Commun. Netw. 2018;2018:1–16. doi:10.1155/2018/7247095.
Yan L-K, Yin H. In: USENIX Security Symposium. DroidScope: seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis; 2012.
Yoo I. Visualizing windows executable viruses using self-organizing maps; 2004. p. 82–9.
Yuan Z, Lu Y, Wang Z, Xue Y. Droid-Sec: deep learning in Android malware detection. ACM SIGCOMM Comput. Commun. Rev. 2014. doi:10.1145/2619239.2631434.
Yuan Z, Lu Y, Xue Y. DroidDetector: Android malware characterization and detection using deep learning. Tsinghua Sci. Technol. 2016;21:114–23. doi:10.1109/TST.2016.7399288.
Zhang S, Xiao X. CSCdroid: accurately detect Android malware via contribution-level-based system call categorization. In: 2017 IEEE Trustcom/BigDataSE/ICESS; 2017. p. 193–200.
Zhao Y, Xu C, Bo B, Feng Y. MalDeep: a deep learning classification framework against malware variants based on texture visualization. Secur. Commun. Netw. 2019;2019:1–11. doi:10.1155/2019/4895984.
Zhong Y, Yamaki H, Takakura H. A malware classification method based on similarity of function structure. In: 2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet; 2012. p. 256–61.
Zhongyang Y, Xin Z, Mao B, Xie L. DroiDalarm: an all-sided static analysis tool for Android privilege-escalation malware; 2013. p. 353–8.

Anson Pinhero is B.Tech in Computer Science & Engineering from SCMS School of Engineering & Technology, Kerala, India. . He has a strong interest in the field of cyber security, machine learning and data privacy.

Anupama M L received her Bachelor of Technology in Computer Science and Engineering from Mahatma Gandhi University, Kerala, India . She received her Master's degree in Computer Science and Engineering with Information Systems from SCMS School of Engineering & Technology, Kerala, India affiliated to APJ Abdul Kalam Technological University . Her research interests are Malware Detection, Machine Learning and deep learning.

Vinod P. is presently Professor at the Department of Computer Applications at Cochin University of Science & Technology, Cochin, Kerala, India. Prior to this he was Professor in Department of Computer Science & Engg. at SCMS School of Engineering & Technology, Kerala, India. He holds a Ph.D from Department of Computer Engineering, Malaviya National Institute of Technology, Jaipur, India. He was Post Doc at Department of Mathematics, University of Padua, Italy and also a Post Doctoral researcher at Malaviya National Institue of Technology, Jaipur, India. He has more than 70 research articles published in peer reviewed Journals and International Conferences. He is a reviewer of a number of security journals, and has also served as programme committee member in the International Conferences related to Computer and Information security. His current research is involved in the development of malware scanners for mobile applications using machine learning techniques. Vinod's area of interest is Adversarial Machine Learning, Malware Analysis, Context aware privacy preserving Data Mining, and Natural Language Processing.

Aneesh N is B.Tech in Computer Science & Engineering from SCMS School of Engineering & Technology, Ernakulam, Kerala, India. He interests are machine learning, data science, digital marketing and also the managerial sections of the engineering platform.



Abhijith S is B.Tech in Computer Science & Engineering from SCMS School of Engineering & Technology, Ernakulam, Kerala, India. His main research interest are data analytics, machine learning and privacy preserving data mining.

Ananthakrishnan S is B.Tech in Computer Science & Engineering from SCMS School of Engineering & Technology, Ernakulam, Kerala, India. He has a strong interest in the field of data privacy, machine learning and malware detection.

Corrado Aaron Visaggio is an associate professor at the Department of Engineering of the University of Sannio, where he teaches "Security of Networks and Software Systems" at the M.Sc. in Computer Engineering. He obtained the M.Sc. in Electronic Engineering (2001) from Politecnico di Bari, and the PhD in Information Engineering (2005) from University of Sannio. His main research interests are: malware analysis, data protection, data protection, threat intelligence. He teaches in Master Programs of Cybersecurity of University of Rome "Tor Vergata", and the International School against organized crime organized by the Italian Ministry of Interior for the education of International Law Enforcement Agencies, and has been instructor at the Department of Intelligence, at the italian Ministry of Interior. He is director of the the Department of Engineering of University of Sannio. He is the scientific leader of several research projects in Cybersecurity, funded by Private and Public Organizations. He collaborates with several Universities (ETH Zurich, University of San Jose, University of Castilla-La-Mancha, University of Lugano, University College Dublin, University of Delft, and SCMS School of Engineering & Technology). He has authored almost one hundred scientific papers and he serves in several Editorial Boards of International journals and Program Committes of international conferences. He is among the founders of the SER&Practice software house.