Personal and Ubiquitous Computing (2024) 28:153–169
https://doi.org/10.1007/s00779-021-01581-w

ORIGINAL PAPER

Malware visualization and detection using DenseNets

V. Anandhi[1] · P. Vinod[2] · Varun G. Menon[1]

Received: 13 March 2021 / Accepted: 28 May 2021 / Published online: 1 July 2021

Abstract
Rapid advancement in the sophistication of malware has posed a serious impact on the device connected over the Internet. Malware writing is driven by economic benefits; thus, an alarming increase in malware variants is witnessed. Recently, a large volume of malware attacks are reported on Internet of Things (IoT) networks; as these devices are exposed to insecure segments, further IoT devices reported have hardcoded credentials. To combat malware attacks on mobile devices and desktops, deep learning-based detection approaches have been attempted to detect malware variants. The existing solutions require large computational overhead and also have limited accuracy. In this paper, we visualize malware as Markov images to preserve semantic information of consecutive pixels. We further extract textures from Markov images using Gabor filter (named as Gabor images), and subsequently develop models using VGG-3 and Densely Connected Network (DenseNet). To encourage real-time malware detection and classification, we fine-tune Densely Connected Network. These models are trained and evaluated on two datasets namely Malimg and BIG2015. In our experimental evaluations, we found that DenseNet identifies Malimg and BIG2015 samples with accuracies of 99.94% and 98.98%, respectively. Additionally, the performance of our proposed method in classifying malware files to their respective families is superior compared to the state-of-the approach calibrated using prediction time, F1-score, and accuracy.

Keywords Convolutional neural networks · DenseNet · Feature maps · Malware visualization · Texture

1 Introduction

In recent years, desktops and smart devices are exposed to serious threat due to the presence of malware attacks [1]. Malware or malicious software are evolving at a faster rate [2, 3]; they are designed to disrupt, gain unauthorized access, and exfiltrate sensitive information from computer systems. A primary motivation for developing new malware is the financial gain associated with it. Hence, it is an industry worth millions of dollars which is increasing every year. According to statistics, data breaches have increased substantially by 40% [4]. Additionally, AV-Test threat report registers [5] more than 350,000 new malware every day, and malware circulation has increased to 114,530 million in 2021; surprisingly, January 2021 alone reported the presence of 607 million malware.

Recently, malware attacks on IoT devices are increasing at an alarming rate. IoT devices have very specific functionalities such as smart healthcare (for monitoring glucose, smart pacemakers, etc), temperature monitoring particularly used in industrial control systems, smart appliances (e.g., smart refrigerator), baby monitoring systems, surveillance system using security cameras, etc. Vulnerable IoT devices of individuals and organizations are largely attacked by hackers primarily due to (a) hardcoded credentials, (b) outdated operating systems, device drivers, and (c) connection of IoT devices to an insecure network and poor web services. All these aforesaid issues transform IoT devices as a pivot to the internal network and expose them to adversary controlled servers. A widely used

✉ P. Vinod
vinod.p@cusat.ac.in

V. Anandhi
anandhi@scmsgroup.org

Varun G. Menon
varunmenon@scmsgroup.org

[1] Department of Computer Science and Engineering, SCMS School of Engineering and Technology, APJ Abdul Kalam Technological University, Thiruvanthapuram, Kerala, India

[2] Department of Computer Applications, Cochin University of Science and Technology, Cochin, Kerala, India

technique for detecting malware is based on "signatures", which is a short string of bytes that uniquely identify known malware. To detect malware, a generated signature of a file is compared with a collection of signatures stored in the database. While this approach is simple, signature detection fails to identify new malware samples, as it is highly sensitive to minor software modification, besides demand human expertise for its creation.

In general, malware detection techniques are classified as static [6] and dynamic [7] approaches. In the static method, analysis is performed on disassembled code to extract opcode sequence, API sequence, functional call graph, strings, etc. Static analysis can quickly derive program semantics but is defeated by source code obfuscation and encryption. The dynamic analysis, also known as behavioral analysis, executes the application in a controlled environment (known as a sandbox) to extract system calls or network traces, and is effective in identifying camouflaged behavior. This approach is robust but is slow and resource-intensive. Besides, malware exhibits restrained behavior on identifying analysis environments.

Generally, the performance of traditional machine learning algorithms depends on robust feature extraction [8, 9] methods. This resulted in the popularity of machine learning in the security domain, especially malware detection [8, 10, 11]. However, these approaches are not effective in detecting new variants or unknown malwares [12]. Recent research has demonstrated the popularity of deep learning techniques in numerous applications such as object detection, image segmentation, medical image analysis, and fraud detection. Hazra et al. [13] exhaustively reviewed diverse deep learning methods and prioritized Convolutional Neural Network, Recurrent Neural Network, Long Short-Term Memory, and Deep Belief Network models as the most promising architectures based on its acceptance in modern applications. Furthermore, deep learning techniques, in particular Convolutional Neural Network (CNN), gained importance in detecting malicious software [14]. Consequently, many intelligent anti-malware solutions with deep learning methods [15–18] have been reported.

A vast majority of malware samples have structural dissimilarity having certain functionality in common. The collection of such malicious programs is known as "family" [10]. In this scenario, the detection of different types of malware is a major concern to both researchers and the anti-virus industry. It is reported that less than 2% code difference exists in newly generated variants. To minimize the domain knowledge and feature extraction cost, researchers have addressed malware detection problems as image visualization techniques, largely due to the presence of visual similarity among variants of the same family. In this approach, a malicious binary is transformed into a two-dimensional image [10], and these images are presented to the machine learning or deep learning classifiers for extracting relevant features for classification. In this way, malware scanners will be agnostic to the operating systems and file types. However, simply representing malware binaries as 2D images is disregarded as they lack semantic information. Additionally, training algorithms on the entire pixels of an image is computationally expensive. Thus, such malware scanners will be impractical to deploy on the devices.

To address the above-stated limitations, we created a deep learning-based malware classifier capable of detecting and classifying malicious executables in real-time. Due to resource constraint in IoT devices, we perform the analysis of malicious samples in the cloud environment. The system used by the end-user periodically transfers executable to our analysis environment deployed on the cloud, which performs data prepossessing and prediction of new samples using deep learning models. Our approach maps executables as grayscale and color images and textures are extracted using Gabor filter, assuming a prevalence of code reuse in new variants. We call such images Gabor images. Subsequently, we create classification models trained on Gabor images using a deep convolutional network and DenseNet. Finally, we prove through comprehensive experiments that DenseNet can accurately detect malicious executables and classify the samples into their respective family in real-time.

In summary, the main contributions of this paper are as follows:

- We develop a malware detection and classification system using different types of visual features (grayscale, RGB, and Markov image). This is achieved by training fine-tuned Convolutional Neural Network (specifically VGG3) and Densenet over two benchmark malware datasets, i.e., Malimg and BIG 2015.
- We conduct comprehensive experiments on image texture and show significant improvement in the performance of deep learning models comparing standard visual features (gray and RGB images).
- We compare our proposed solution with the state-of-the-art approaches using the identical datasets for a fair comparison. Furthermore, through intensive experiments, we show that our approach surpasses the previous studies both in execution speed and classification results. In addition, we created adversarial samples by injecting additive noise into files. Finally, we observed a marginal drop in the performance of DenseNet. This proves the efficacy of the trained model for detecting new samples.

Section 2 gives a brief introduction to the related work. Our contributions right from the preprocessing to deep learning classification methods are described in Section 3.

Section 4 explains the evaluation metrics used in this paper. The experimental results and discussions are shown in Section 5. Section 6 concludes the paper with directions for future work.

2 Related work

In this section, we present the related research papers based on feature analysis and malicious code visualization.

2.1 Feature analysis

The main approaches of analyzing malware based on feature analysis are static and dynamic code analyses. In static analysis, the code is disassembled and examined for malicious patterns. Dynamic analysis is a behavior-based procedure where the code is executed in a virtual environment and the execution trace is analyzed. The static analysis offers the complete analysis procedure but suffers from code obfuscation. Dynamic analysis is more efficient but the main disadvantage is a time-consuming process and resource-consuming too. Some malicious apps might not be observed because the environment is only a virtual one which does not satisfy all conditions of a real one.

The static analysis method analyzes binary values or hex values and extracts some features from the files. Machine learning methods can be applied to the static features obtained. It is observed that this method can achieve better accuracy. Previous studies of static methods use permissions to verify and check application risk, but these methods are difficult to guarantee high accuracy [6, 19]. A method based on static analysis is proposed in [6] for malware detection in Android devices which is performed by extracting specific feature information like serial number, and based on this classify malicious apps of similar apps. Nezhadkamali et al. [20] proposed a permission-based algorithm that uses Intents and API sequences. The method modifies the obtained values using the same features maps. In this way, they were able to obtain accuracy to 98.6%.

Vasan et al. [21] proposed a data mining method that detects malware variants by static methods. The analysis introduced extraction of features and efficiently classifies malware family which solves the data imbalance problem with the help of the augmentation technique.

The dynamic analysis method executes the files in a virtual environment. Shezan et al. [22] proposed a detailed study on awareness to the users about common vulnerabilities present in the OS and the quality tools for testing the vulnerabilities.

Han et al. [7] proposed a method based on correlation with semantics and a combination of static and dynamic API functions and construct an overall feature map vector space for malware detection. In [23], a probability matrix–based classification system and machine learning are developed for static and dynamic analyses using CNN with spatial pyramid pooling techniques, where the accuracy of 98.82% is achieved. Yoo et al. [24] proposed a hybrid model consisting of a Random forest classifier and deep learning models with 12 hidden layers to identify malware and benign files, and obtained a detection rate of 85.1%. In short, to detect malicious code a behavior model is established. These behavior models achieved better detection results but developed unreliable results. Moreover, dynamic analysis consumes more processing time as the computational overhead is more, which leads to less accuracy when large datasets are involved.

2.2 Malicious code visualization

Nowadays, many tools can visualize the available data. Several studies propose malware classification using visualization techniques. The fundamental idea employed in these approaches is to dissect the patterns of malicious apps and new malware variants. It also identifies the changes made on benign software due to viruses. The analysis also finds the relation between different malware families and can identify the unknown malware family too. Recent studies point out that the patterns of benign and malicious apps are different [25]. Experiments performed using CNN in this paper report an accuracy of 90.67%. Furthermore, this paper concludes the existence of relevant features for classifying samples into their respective family.

Nataraj et al. [10] proposed a novel method for malware detection, where the binary executables packed in families are converted into grayscale images of fixed size based on the technique of texture analysis. Then, the features are used for the detection of malware and the efficiency was better in less time.

Han et al. [26] proposed a dynamic analysis visualization procedure to obtain RGB-colored pixels using the opcode values from malware files and the similarities are calculated and the procedure applies to packed malware samples too.

Zhong and Gu [12] proposed a malware detection method on deep learning by using multiple levels, where each model does not work on the entire dataset, but it works on part of the dataset for a group of malware families so that each model can learn a particular data distribution. Finally, all the models together decide hierarchically and make a final decision on the data.

Due to the rapid growth of malware variants, a classification method based on Markov images and deep learning [27] was introduced using byte-level. It converts the binaries into Markov images on the two standard malware datasets, the Microsoft dataset and the Drebin

dataset and the average accuracy rates obtained are 99.264% and 97.364% respectively.

Deep learning has been a solution in many malware classification applications. In [28], a deep learning model classifier was developed for CNN-based architecture to classify malicious samples which achieved accuracies of 98.52% and 99.97% on the Microsoft datasets and Malimg datasets respectively. Roseline et al. [29] proposed a system that does not require backpropagation or hyperparameter tuning, thereby reducing model complexity. The model has detection rates of 97.2% and 98.65% for BIG 2015 and Malimg datasets respectively. Their model deficits in identifying unknown samples of untrained families.

Gibert et al. [30] developed a deep learning system that learns visual features of Malimg and BIG2015 malicious executables to classify them into families. To visualize, each malware sample was converted to a grayscale image. A fine-tuned CNN model was developed which classified malicious files of both the datasets with accuracies of 98.48% and 97.49% respectively. Executables of Microsoft BIG2015 were transformed to opcodes and finally represented as images [31]. Then, GoogleNet model and ResNet model were applied to detect malicious files. The ResNet attained 88.36% precision, whereas 74.5% accuracy was obtained on GoogleNet.

Hamad Naeem et al. [32] developed a hybrid approach on color images to detect IoT malware using two datasets. A deep CNN model was selected to extract visual features. The detection accuracy was reported on two datasets as 97.81% and 98.47% respectively. Recently, Turker et al. [33] proposed a cognitive and intelligent anti-malware model for classifying samples of the Malimg dataset. They created hybrid features, extracted through LBP-SVD-LPNet. Subsequently, Principal Component Analysis (PCA) was applied to obtain relevant attributes, and finally, a classification model was developed using Linear Discriminant Analysis (LDA). This method achieved 87.62% F1 score and 88.08% detection rate. Çayır et al. [34] proposed Random Capsule Network (RCNF) reducing the variance of different CapsNet. They conducted the experiments on a public benchmark dataset, i.e., Malimg and BIG2015. The authors reported that the RCNF achieves F1 scores of 0.966 and 0.982 on Malimg and BIG2015.

Sun and Qian [35] proposed a method to combine the static analysis with recurrent neural networks and CNN. By using this technique, the accuracy obtained was over 92%. By increasing the training dataset, the accuracy reached 99.5%. Feng et al. [36] proposed an Android malware detection system in a real-time environment on mobile devices and found that memory usage was 60 MB and the execution time was 0.46 s. Detailed analyses on accuracy and performance on various mobile devices were performed and accuracy obtained was 96.75% on all devices with a response time of fewer than 3 s.

To detect and classify malware samples in real-time, we propose the implementation of DenseNet, trained on visual artifacts of the malicious sample. Another important aspect of our proposed approach is the use of texture-based images for representing executables. This reduces computational overhead by only allowing the classifier to remember prominent patches of the image which acts as its global feature.

3 Proposed method

Malware writers usually change a part of the previous codes available to produce new malware [37] variants. If we try to represent malware as an image, these changes can be detected by robust malware visualization techniques. With this idea, we visualize malware binary files as grayscale images, RGB images, and Markov images. Later, we perform texture analysis to determine specific components around the region of interest.

Figure 1 shows the overall architecture. In this system, the benign and malware files are represented as fixed size images. The benign executables were collected from various sources which include laptops and the Internet [38–41]. Then, we use VirusTotal to check if the collected executables are malware free. In addition, public benchmarks, i.e., Malimg [10] and BIG2015 [42], are considered as malware datasets. We choose to study on these samples as (a) both the datasets have class imbalance, this will help us evaluate the robustness of the proposed model, and (b) the samples in the aforementioned dataset are exhaustively evaluated in a large collection of prior work [12, 27, 30, 37, 43]. All binaries including malware and goodware were converted to images. Subsequently, these image files were supplied to DenseNet which learns appropriate features necessary for the task of classification.

3.1 Preprocessing

Our system transforms executables to grayscale, RGB, and Markov images. The steps for converting a given binary into an image are explained in the subsequent paragraphs.

3.1.1 Grayscale images

The given hexadecimal values of executables are converted into binary. We read bytes and express it as gray values in the range of 0–255, where 0 indicates black and 255 represent white. These decimal values are organized as a two-dimensional matrix of size 256×256.

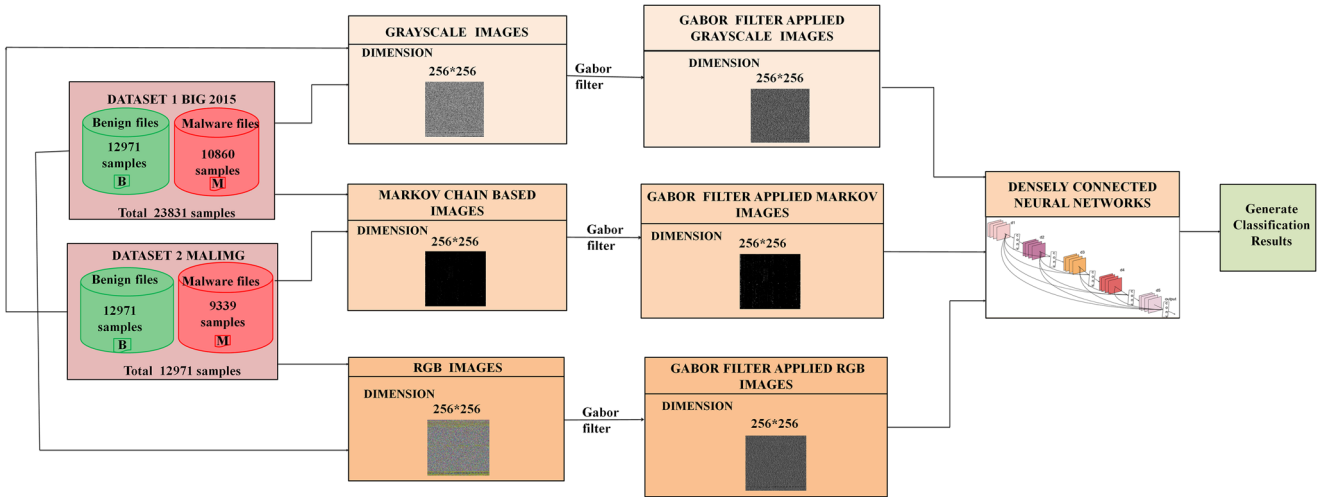


Fig. 1 Overall architecture of malware detection method

3.1.2 RGB images

In this process of visualization, we represent malware and legitimate binaries as RGB channels. To accomplish this task, we create a color map, a two-dimensional data structure having 256 rows and 256 columns, consisting of pixel values. For converting an executable to an RGB image, we read two bytes at a time, the first byte is used as row index, and the second byte as column index of the color map. Finally, the element at the intersection of the row and column indices is chosen as a pixel in the red, green, and blue plane.

3.1.3 Markov images

Figure 2 shows the steps for converting binaries into Markov images. Generally, for creating visual features, one byte is read and mapped to an equivalent pixel [10]. The influence of truncating and padding bytes to a file on classifier accuracy is not precisely addressed in prior studies. To circumvent the above-stated issue, Markov images are developed, where the bytes are considered as a stochastic process. This means that probability of byte b_i is related to byte b_{i-1}; thus, the Markov chain is represented as:

$$P(b_{i+1}|b_0 \cdots b_i) = P(b_{i+1}|b_i)$$

The steps to generate Markov images are summarized below:

- **Step 1:** First a frequency matrix is created which records the occurrence of consecutive pixel pairs denoted by $f(x, y)$, for bytes x and y.
- **Step 2:** Using the frequency matrix, we create a Markov probability matrix where the elements are transfer probabilities of byte x and y. Probabilities in the Markov probability matrix (see (1)) are populated by taking each element in the frequency matrix and dividing by the corresponding row sum.
- **Step 3:** Finally, we map the probability to pixels. Here, each element in the probability matrix is multiplied by 255 and divided by the maximum probability to generate the Markov image of size 256×256.

$$P(x, y) = P(y|x) = \frac{f(x, y)}{\sum_{n=0}^{255} f(x, y)}. \quad (1)$$

3.1.4 Gabor images

Gabor filter is popularly used in diverse computer vision applications mainly in object detection, feature extraction, and image segmentation. These filters mimic the mammalian visual cortex to recognize textures in an image. A 2D-Gabor filter derives features in both the spatial and

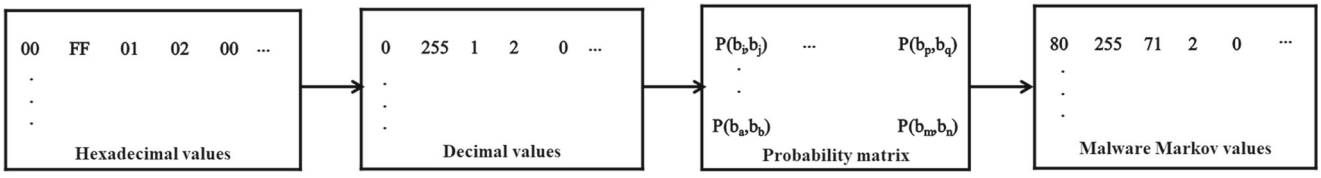


Fig. 2 Generation of Markov images

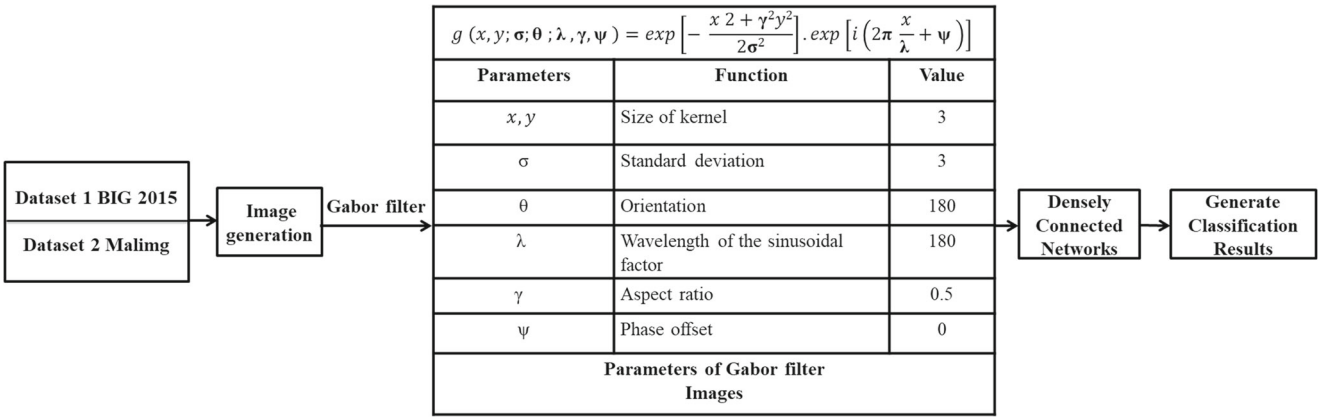


Fig. 3 Parameters of Gabor filter applied to images

frequency domains. A Gabor function is the product of sinusoidal and Gaussian functions in the time domain, and convolution of transforms of these functions in the frequency domain (refer to Fig. 3). Gabor filter analyzes whether there is a particular frequency of information in that particular direction around the region of analysis. We can obtain a bank of Gabor filters by varying the frequencies and orientations in the Gabor function. The parameters at which we obtained better results are also shown in Fig. 3. An input image is convoluted through a set of Gabor kernels to obtain image texture. The figure briefly presents the steps for generating Gabor images. Later, Gabor images are fed to CNN and DenseNet for classification and prediction.

3.2 Feature extraction and learning

Recently, researchers have attempted using deep learning techniques for classifying and detecting malicious files. As convolutional neural networks (CNN) and densely connected networks (DenseNet) have proven to be very efficient for computer vision problems [44, 45], we thus adopt aforesaid architectures for malware classification using visual features. In the following subsections, we introduce CNN and DenseNet.

3.2.1 Convolutional neural networks

Convolutional neural networks have always become the popular machine learning algorithm [14] for image classification and object detection. CNNs have two important operations: (a) convolution and (b) pool. In the forward pass, the convolutional layer utilizes a bank of filters, each of which convolves across the width and height of the executables image to output feature maps. Next, the max pooling layer progressively downsamples the feature map. Max pooling takes the maximum value by sliding 2×2 window over the feature map. The combination of convolution followed by max-pool is a new image that is further presented to subsequent layers of CNN. Finally, the output of the max-pool is flattened and given as input to the fully connected network which is further connected to the output layer. The class of malware file is given to the output through softmax activation function for malware classification and sigmoid activation function for the malware detection task. During the training phase, the values of the parameters are updated continuously until the learning is complete. In our study, we use the deep convolutional network, in particular VGG3 [46]. VGG3 learns representative attributes of a sample without the need for manual feature extraction by performing multiple nonlinear transformation to produce abstract representation for each executable image.

CNN has found more importance in medical applications like chest disease detection [47, 48]. Choudhary and Hazra employed the VGG16 model to train the medical images and the results obtained are 98% of training accuracy and 97% of test accuracy.

3.2.2 Densely connected networks

The motivation for selecting DenseNet is circumventing the vanishing gradient problem. For deeper networks, gradients are not propagated back to the previous layers or the initial layers. The gradient shrinks for a deeper network due to which the initial layers do not learn. If there is no significant change in the gradient, it is understood that the weights are not updated or in effect no learning takes place. Hence to resolve the above-stated issue we have considered DenseNets. DenseNets have interconnection paths where features are combined by concatenation. The main highlights of the DenseNet architecture are feature reuse, vanishing gradient problem, feature propagation, and less parameter count [44]. These interconnections form deep and dense paths that learn better than other networks.

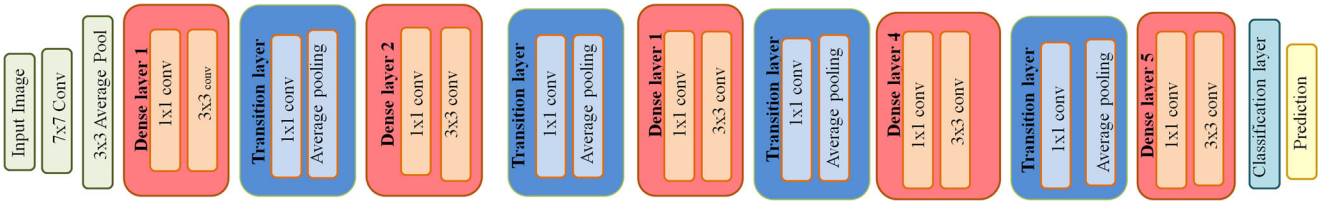


Fig. 4 Detail architecture of DenseNet

The DenseNet comprises dense blocks, composite function, and transition layer. First is the dense block where each block comprises n dense layers. These dense layers are connected such that each layer receives information from all preceding layers and passes its information to all subsequent layers. Thus, each layer receives collective information from all the subsequent layers. The width and height of the feature maps stay the same in a particular block. Each layer consists of two convolutional operations: 1×1 convolution (CONV) for extracting features and 3×3 CONV for bringing down the feature count. The composite function consists of batch normalization, Rectified Linear Unit (RELU), and 3×3 convolution layer. This function concatenates the output of the preceding layers to the next layer.

Finally, the transition layer consists of 1×1 CONV operation which reduces the parameter count, followed by the 2×2 average pool layer, responsible for downsampling the dimension feature space. Figure 4 illustrates the detailed architecture of DenseNets. The whole idea of DenseNet is to reuse features. Additionally, maximize the flow of information between dense layers. As the model requires only a fewer number of layers, they are easy to train and a lesser number of parameters need to be learned. In our study, we used a specific variant of DenseNet, i.e., DenseNet201, consisting of 201 layers deep. The motivation of using DenseNet201 was drawn due to improved performance as reported in [44]. Additionally, DenseNet has a strong gradient flow. The error signals are transferred to earlier layers compared to the conventional deep neural network. Thus, the earlier layers timely administer errors without waiting longer for gradients, computed from previous layers.

Table 1 Number of trainable parameters in the proposed DenseNet model

Hyperparameters used in the model	Value
No. of layers in DenseNet	201
Learning rate	0.001/0.01
No. of epochs	30/50
Batch size	32/64

In our experiments, we have used DenseNet architecture with the number of trainable parameters as shown in Table 1.

4 Evaluation metrics

The evaluation metrics used in our work for the classification and malware detection are Precision, Recall, F1 measure, and Accuracy. These metrics are calculated using True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). TP indicates the number of malware samples correctly identified as malware. TN indicates the number of benign samples accurately identified as legitimate. FP is the number of benign samples misclassified as malware. FN indicates the number of wrongly malware images. The evaluation metrics are defined using (2) through (5).

$$Precision(P) = \frac{TP}{TP + FP}, \tag{2}$$

$$Recall(R) = \frac{TP}{TP + FN}, \tag{3}$$

$$F1measure(F) = 2 * \frac{R * P}{R + P}, \tag{4}$$

$$Accuracy(A) = \frac{TP + TN}{TP + TN + FP + FN}. \tag{5}$$

The performance of DenseNet on different types of images is compared and shown in Figs. 5, 6, 7, and 8. An ideal malware detection model should have a high value of F1 measure. For the Gabor filter applied to Markov images, F1 measures obtained are 98.73% and 99.94% for BIG 2015 and Malimg datasets, respectively.

5 Results and discussions

Implementation and setup The hardware requirements for conducting the experiments are Intel(R) Xeon(R) CPU @ 2.20GHz*2 processor with 12 GB RAM capacity and 1 TB hard drive. Experiments were conducted on Ubuntu 18.04 platform. Instead of normal CPU computing, the use of NVIDIA Tesla P100 16 GB graphics card increases

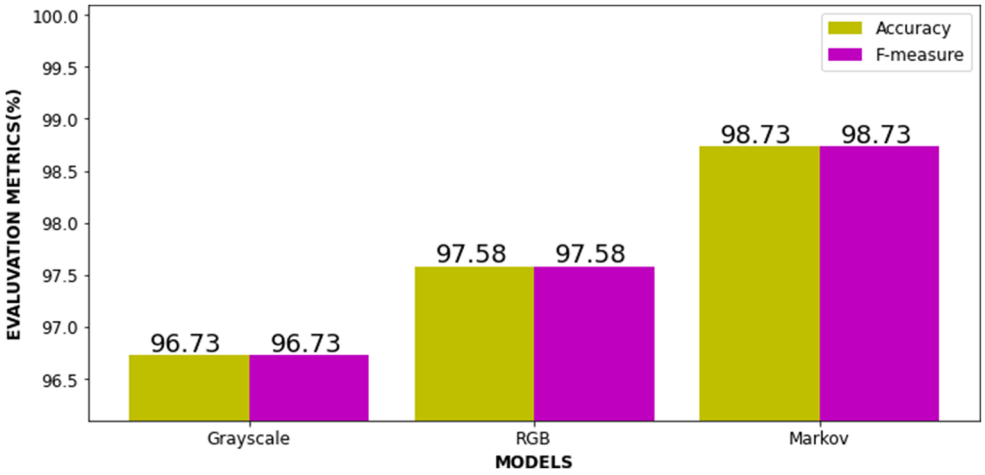


Fig. 5 Performance measures of DenseNet without applying Gabor filter, Dataset Benign + BIG 2015

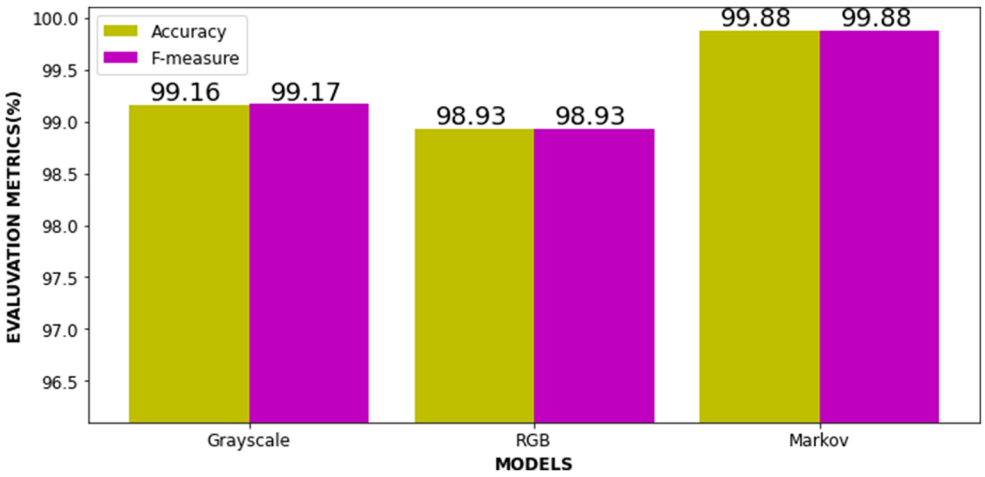


Fig. 6 Performance measures of DenseNet without applying Gabor filter, Dataset Benign + Malimg

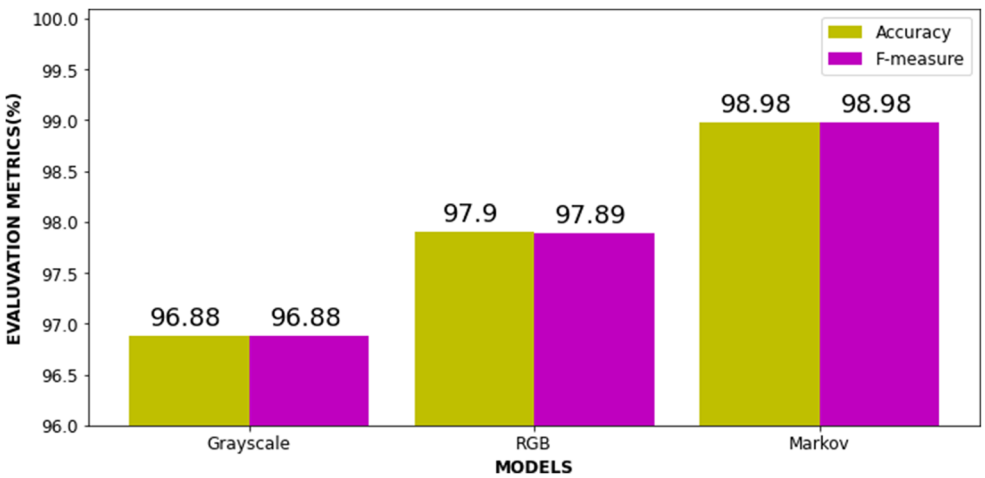


Fig. 7 Performance measures of DenseNet using Gabor filter applied on 256×256 images, Dataset Benign + BIG 2015

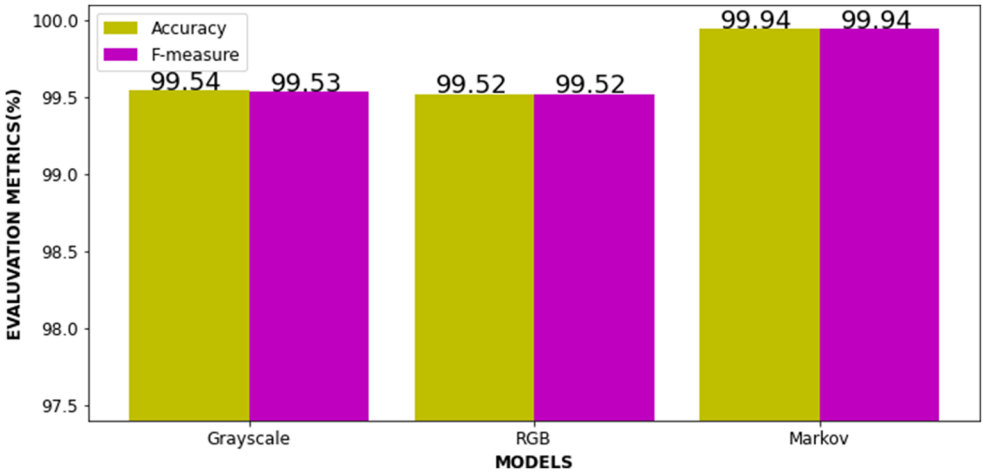


Fig. 8 Performance measures of DenseNet using Gabor filter applied on 256×256 images, Gabor filter applied images, Dataset Benign + Malimg

the computational speed. The software implementation was done using Python 3.7.6 programming language, with packages included such as TensorFlow 2.2.0 and Keras 2.3.1 with Scikit-learn machine-learning library and Matplotlib for visualizing results.

Dataset The experiments are performed on two public benchmark malware datasets, BIG 2015 [42] and Malimg [10]. Malimg dataset has 9339 samples grouped in 25 families; all binaries are converted to grayscale images and published for use. BIG2015 dataset has been released as Kaggle competition; this dataset contains 10868 ASM and byte files corresponding to 9 different malware families. The benign dataset samples constituted of executable collected from [38–41]; furthermore, each executable was checked for malicious files by uploading the file to VirusTotal.

For conducting experiments, we created two datasets. Dataset 1 comprises 23,831 samples where 12,971 samples were benign and 10,868 BIG 2015 samples. Out of the 12,971 benign samples, 9080 samples were used as train set and 3891 samples were used as a test set. From the BIG 2015 malware dataset, 7602 samples constituted the train set and 3258 samples were reserved as the test set. In addition, we created Dataset 2 having the same number of the benign sample as in Dataset 1, but has 9339 Malimg images. From the Malimg malware dataset, 6537 samples were used as a train set and the remaining 2802 files were used for prediction. All images were resized to 256×256 for CNN and DenseNet. We conduct the following experiments:

- **Experiment-1** Effect of image representations on classification performance.
- **Experiment-2** Performance of DenseNet in detecting executables.
- **Experiment-3** Comparison of VGG3 and DenseNet in predicting malicious executables.
- **Experiment-4** Classification of malware samples into their respective families (i.e., multi-class classification).
- **Experiment-5** Comparative analysis of proposed model and state-of-the-art malware detection/classification methods.
- **Experiment-6** Performance evaluation of DenseNet model on evasion attack.

5.1 Effect of image representations on classification performance

In this section, we compare classifier results obtained while inputting different types of images. Table 2 shows the results of the prediction obtained for two datasets. An F1-measure of 98.73% is achieved for BIG2015 samples, and 99.88% F1-measure is obtained for Malimg samples during prediction. The best classifier performance is obtained with Markov images, followed by RGB, and inadequate results are attained for grayscale images. Additionally, improved performance is obtained if textures are extracted from images and presented during the training phase. We notice an increase in F1-measure for BIG2015 Gabor images (grayscale, RGB, and Markov); improvements in F1-measure attained are 96.88%, 97.81%, and 98.98% respectively. Similarly, F1-measures achieved on Malimg dataset are 99.45%, 99.52%, and 99.94% respectively.

In all cases, we can visualize that the classifier performance is limited when trained on grayscale images. Furthermore, the results of grayscale BIG2015 images are less compared to Malimg. Especially in the case of BIG2015, executables were disassembled using IDAPro, and such tools generate output with a fixed structure. In particular, we observed a large volume of black pixels in the grayscale images. This would cause the max pool layer to always output identical values, and such dark regions do not contribute meaningful information during the learning phase. Nevertheless, we also witness that deeper network structures especially VGG3 tend to output moderate results and tend to capture the localized association between the

Table 2 Performance evaluation of convolution neural network and DenseNets without and with using Gabor filters

			Without Gabor filter		With Gabor filter	
Dataset	Image	Architecture	A (%)	F (%)	A (%)	F (%)
		VGG3 Dropout	96.37	96.38	96.39	96.39
	Grayscale	DenseNet	96.73	96.73	96.88	96.88
		VGG3 Baseline	97.38	97.38	97.89	97.89
	RGB	DenseNet	97.58	97.58	97.90	97.89
		VGG3 Dropout with Batch normalization 97.97	97.96	98.81	98.81	
BIG 2015	Markov	DenseNet	98.73	98.73	98.98	98.98
		VGG3 Dropout with Batch normalization	98.72	98.72	99.45	99.45
	Grayscale	DenseNet	99.16	99.17	99.54	99.17
		VGG3 Baseline	97.83	97.84	99.21	99.21
	RGB	DenseNet	98.93	98.93	99.52	99.52
		VGG3 Dropout with Batch normalization	99.16	99.16	99.82	99.82
Malimg	Markov	DenseNet	99.88	99.88	99.94	99.94

pixels. However, superior outcomes are attained in the case of Markov images. Primarily, extraction of texture improves detection, as such images carry minute details such as dots, lines, and boundaries separating different regions in the image. Consequently, the best results are achieved with Markov-Gabor filter images. In addition, we observe moderate accuracy with classifiers trained on RGB images, although RGB images enhance changes in sub-region relatively better compared to the grayscale images. Hence, such differences in regions are easily captured by the deep classifiers during the training phase, and eventually, distinguish malware from their families.

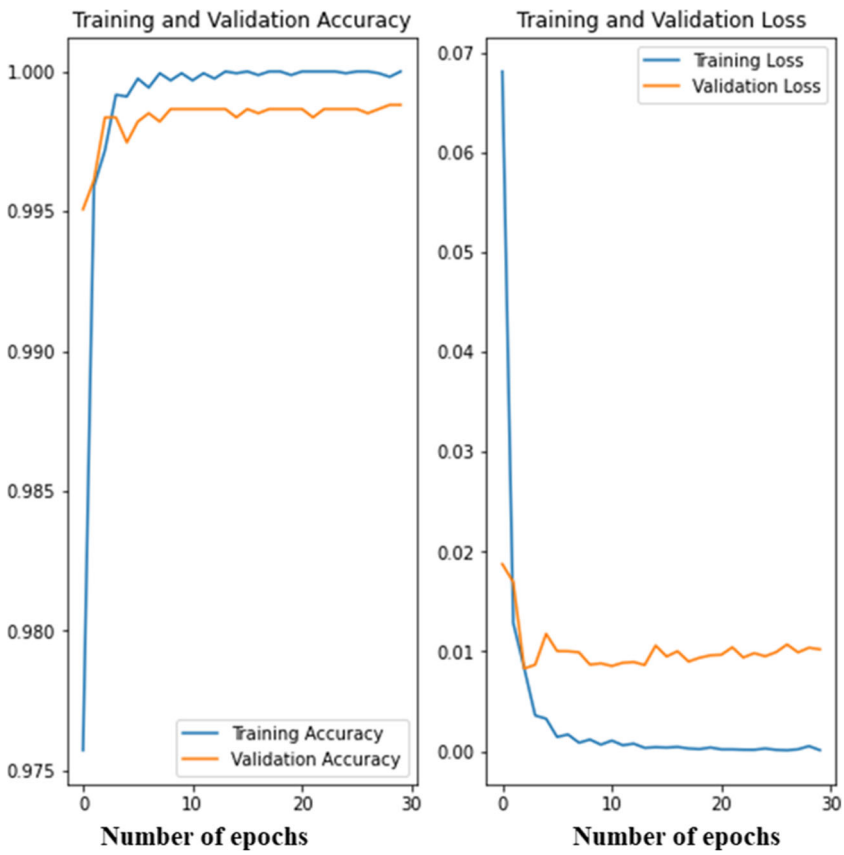


Fig. 9 Comparison of training and validation loss and accuracy with validation and testing samples

5.2 Performance of DenseNet in detecting executables

From Figs. 5, 6, 7, and 8, it is clear that F1-scores of 99.94% and 98.88% are attained for Malimg and BIG2015 respectively. Comparing all figures, it can be seen that highest F1-score and accuracy are obtained if Gabor filter applied Markov images were used to train the DenseNet. Figure 9 depicts the accuracies on the train and validation sets. Likewise, the validation loss on the dataset is reported in the same figure. It is evident that the accuracy improves beyond 10 epochs and converges between 25 and 30 epochs. An identical trend is obtained while we evaluate the loss. In particular, beyond 25 epochs convergence in the loss can be witnessed. Furthermore, in Table 3, we compare time for predicting new samples along with hyperparameters for different DenseNet architectures. We can visualize from Table 3 that the average execution time for predicting a new sample in both datasets is in the range of 0.004 to 0.006 ms which matches real-time scanning of files by antivirus software.

From all experiments, we see that DenseNet obtains a very high F1-score and accuracy particularly due to two fundamental reasons: (a) deeper architecture and (b) interconnections with the previous layers. The deeper network advocates each layer to extract executable image pattern which is disparate from other layers. Besides, increased inter-connectivity of layers allows the re-use of feature maps of previous layers and creates a new image representation that carries information from all predecessor layers along with the current layer. This resolves the problems induced with redundant layers which are typical in deeper architectures.

5.3 Comparison of VGG3 and DenseNet in predicting malicious executables

Table 2 compares the performance of VGG3 and DenseNet on different types of images. Here, we see that for BIG 2015 dataset represented in the form of Gabor images; VGG3 shows a maximum F1-score of 98.81%. While for the identical images, DenseNet attains F1-score of 98.98% which is 0.17% higher than VGG3. Similar trends in the results were observed for Malimg samples. It can be observed that DenseNet accuracy and F1-score for Gabor images is 99.94%, while for VGG3 an F1-score of 99.82% was obtained. Even though we found marginal improvement in the DenseNet performance, the prediction time for BIG 2015 with this model is 0.006ms, and 0.004ms for Malimg files. In addition, we also found a reduction in the number of parameters for implementing DenseNet architecture comparing the VGG3 models.

Table 3 Network topology and performance metrics for image dimension 256×256 by applying Gabor filter with detection time

Image type	Benign + BIG 2015			Malimg			
	Network topology	Hyper-parameters	Prediction time (ms)	Network topology	Hyper-parameters	Prediction time (ms)
Grayscale	DenseNet201+ Dense(1024)+ Dropout(0.1)+ Dense(2048)+ Dropout(0.1)+ Dense(4096)+ Dropout(0.1)	lr=0.01, epochs=50, batch_size=64	Total time= 27.107, Avg time= 0.004	DenseNet201+ Dense(1024)+ Dense(2048)+ Dense(4096)	lr=0.001, epochs=30, batch_size=32	Total time= 25.581, Avg time= 0.004
RGB		lr=0.01, epochs=50, batch_size=64	Total time= 39.871, Avg time= 0.006		lr=0.001, epochs=30, batch_size=32	Total time= 25.487, Avg time= 0.004
Markov		lr=0.01, epochs=50, batch_size=64	Total time= 40.857, Avg time= 0.006	lr=0.001, epochs=30, batch_size=32	Total time= 25.671, Avg time= 0.004	

Summary From experiments 1, 2, and 3, we conclude that different deep learning models trained with Gabor Markov images accurately represented malware and benign executables comparing other types of images (grayscale and RGB). Classification results (both F1-score and accuracy) are superior for DenseNet while comparing different CNN models used in this study. In addition, the small prediction time along with a fewer number of parameters required to model DenseNet comparing VGG3 shows its efficacy in detecting malware executables from goodware.

5.4 Classification of malware samples into their respective families (i.e., multi-class classification)

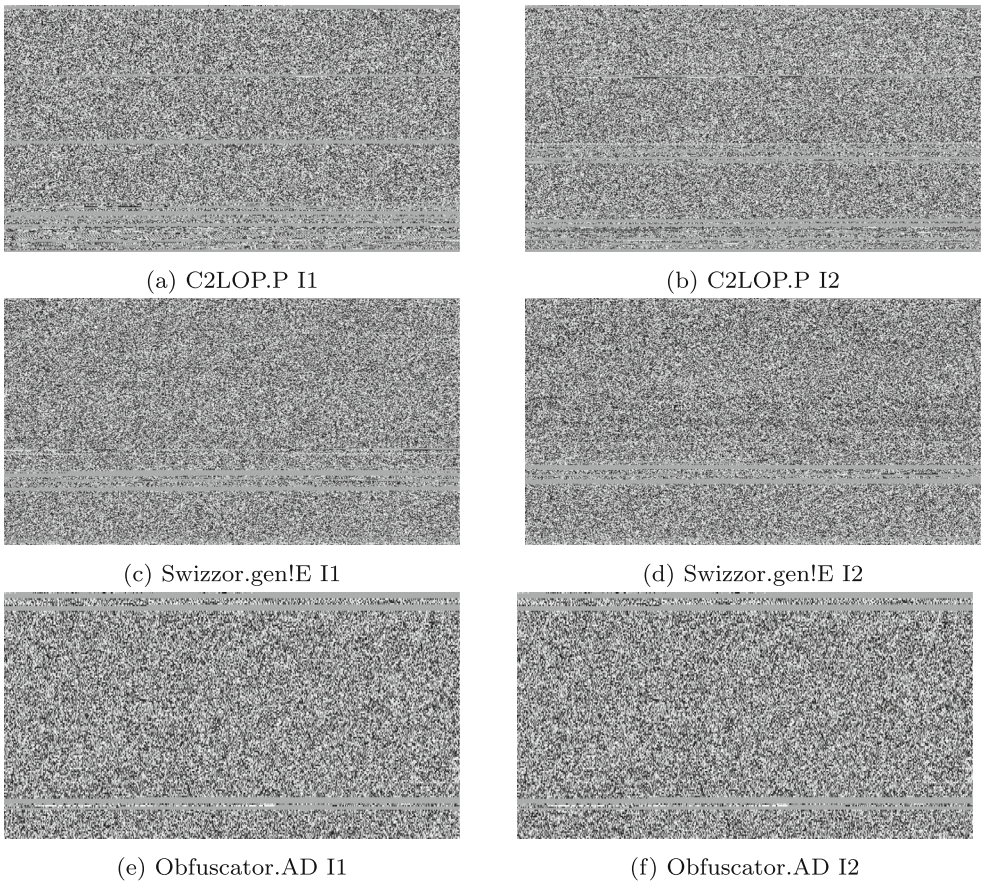
We illustrate the results of characterizing malware families on different categories of images. These experiments were repeated on two benchmark datasets used earlier in previous experiments. The accuracy obtained using DenseNet is 98.53%. For Malimg executables, an F1-score of 98.88% is attained; refer to Table 4. The confusion matrix of the two datasets is shown in Figs. 10 and 12. From the results, we can see that 16 out of 25 families attained 100% detection, test samples of 6 malware families had an F1-score greater than 95%, and finally, 92.68% of Malex.gen!j, 89.47% of Swissor.gen!E, and 42.5% of Swizzor.gen!l were identified by the trained DenseNet model. In particular, we found that 42% of Swizzor.gen!l variants were wrongly labeled as C2LOP.P (refer to Fig. 12).

Figure 11 compares the visual representation of certain samples belonging to three families which reported the highest misclassification (Fig. 12). As we can see, images of malware samples of these families have identical visual patterns due to similar distribution of bytes. Consequently, the classification models wrongly label samples at the prediction time. Thus, to improve the accuracies and F1-score of the DenseNet, we merged malicious samples of C2LOP and Swizzor to form a single-family during the training phase. Specifically, C2LOP is a member of Swizzor; hence, the malicious samples of these families exhibit similar behaviors [49]. The confusion matrix in Fig. 13 shows that the performance of DenseNet significantly improves. Now, the F1-score of malware

Table 4 Comparison with running time required for Image Dimension 256×256 for BIG 2015 and Malimg datasets

Models	A (%)	F (%)	Prediction time
Prior work			
CNN+1LSTM (BIG 2015)	97.64	94.15	32.0 ms
CNN+2LSTM (BIG 2015) [16]	97.91	95.52	62.0 ms
GRU+SVM (Malimg) [15]	85.00	85.00	23.7 ms
GIST+KNN (Malimg)	92.10	91.70	60.0 ms
GIST+SVM (Malimg)	92.50	91.40	64.0 ms
GLCM+SVM (Malimg)	93.40	93.00	48.0 ms
Bat algorithm (Malimg) [37]	94.60	94.50	20.0 ms
Tuncer et al. [33] (Malimg)	88.08	87.62	–
Cayr et al. [34] (Malimg)	98.72	96.61	–
Vasan et al. [21] (Malimg)	98.82	98.75	0.81 seconds
Kalash et al. [28] (Malimg)	98.52	–	–
Kalash et al. [28] (BIG 2015-Setting-A)	98.99	–	–
Khan et al. [31] (BIG 2015)	88.36	–	9248 seconds
Gibert et al. [30] (Malimg-25 families)	–	94.8	–
Gibert et al. [30] (Malimg-19 families)	–	98.4	–
Gibert et al. [30] (BIG 2015)	97.5	94.0	0.001 seconds
Naeem et al. [32] (Malimg)	–	98.75	30 seconds
Our Proposed method: Classification			
Gabor filter applied DenseNet Markov (BIG 2015)	98.52	98.53	11.5 ms
Gabor filter applied DenseNet Markov (Malimg)	98.97	98.88	5.0 ms
Gabor filter applied DenseNet Markov images (on combining C2LOP and Swizzor families of Malimg)	99.36	99.37	4.95 ms
Our Proposed method: Detection			
Gabor filter applied DenseNet Markov (BIG 2015)	98.98	98.98	5.7 ms
Gabor filter applied DenseNet Markov (Malimg)	99.94	99.94	3.8 ms

Fig. 10 Confusion matrix of BIG 2015



(a) C2LOP.P I1 (b) C2LOP.P I2

(c) Swizzor.gen!E I1 (d) Swizzor.gen!E I2

(e) Obfuscator.AD I1 (f) Obfuscator.AD I2

Fig. 11 Visual similarity in samples of C2LOP, Swizzor.gen!E, and Obfuscator.AD families

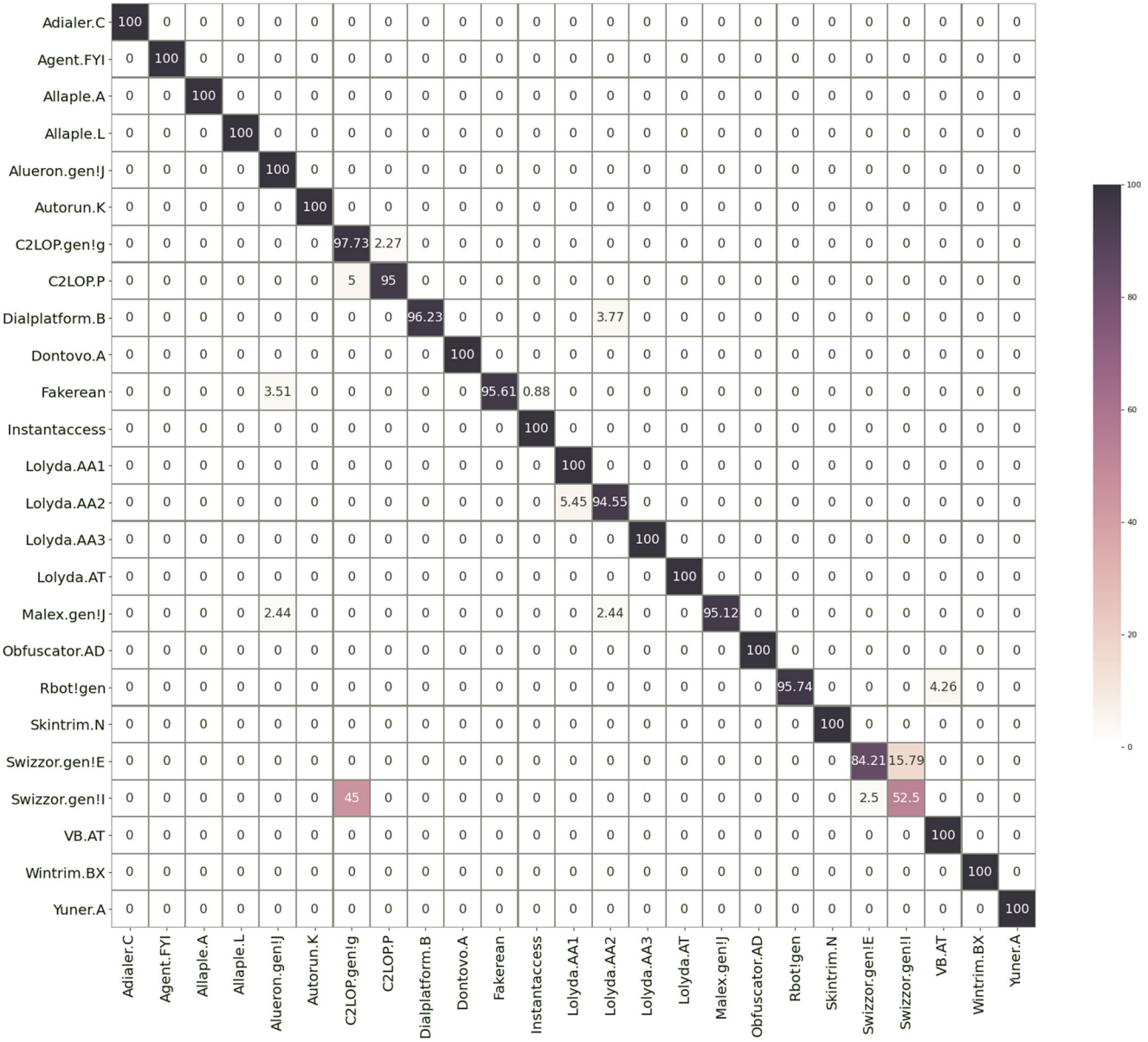


Fig. 12 Confusion matrix of Malimg dataset among variants

families is in the range of 95–100%. In particular, we can find that 100% of samples of C2LOP and Swizzor.gen!E were identified by the classification model.

5.5 Comparative analysis of proposed work with the state-of-art solutions

Comparative analysis is performed based on two different experimental settings: malware detection and classification. In [15], an intelligent malware system is proposed on the Malimg dataset which achieved an accuracy of 85% and a prediction time of 23.7 ms. Quan et al. [16] proposed a deep learning–based malware classification method that requires only the knowledge of identification of features and has obtained the maximum accuracy of 97.91%. In [30], BIG2015 and malimg datasets were used where the visual similarity was considered and the accuracy obtained was 98.4%. Zhihua et al. [37] give a comparison on various gist techniques and the bat algorithm on the malimg dataset and obtained an accuracy of 94.5% with a prediction time of 20 ms.

We compare the outcome of our proposed method with the state-of-the-art solutions. To have fair comparison, we considered only those classification systems which employed identical dataset as we used in our work. Table 4 summarizes the results of prior work and our proposed solution. We can observe that performance of our solutions is better compared to all other works listed in Table 4

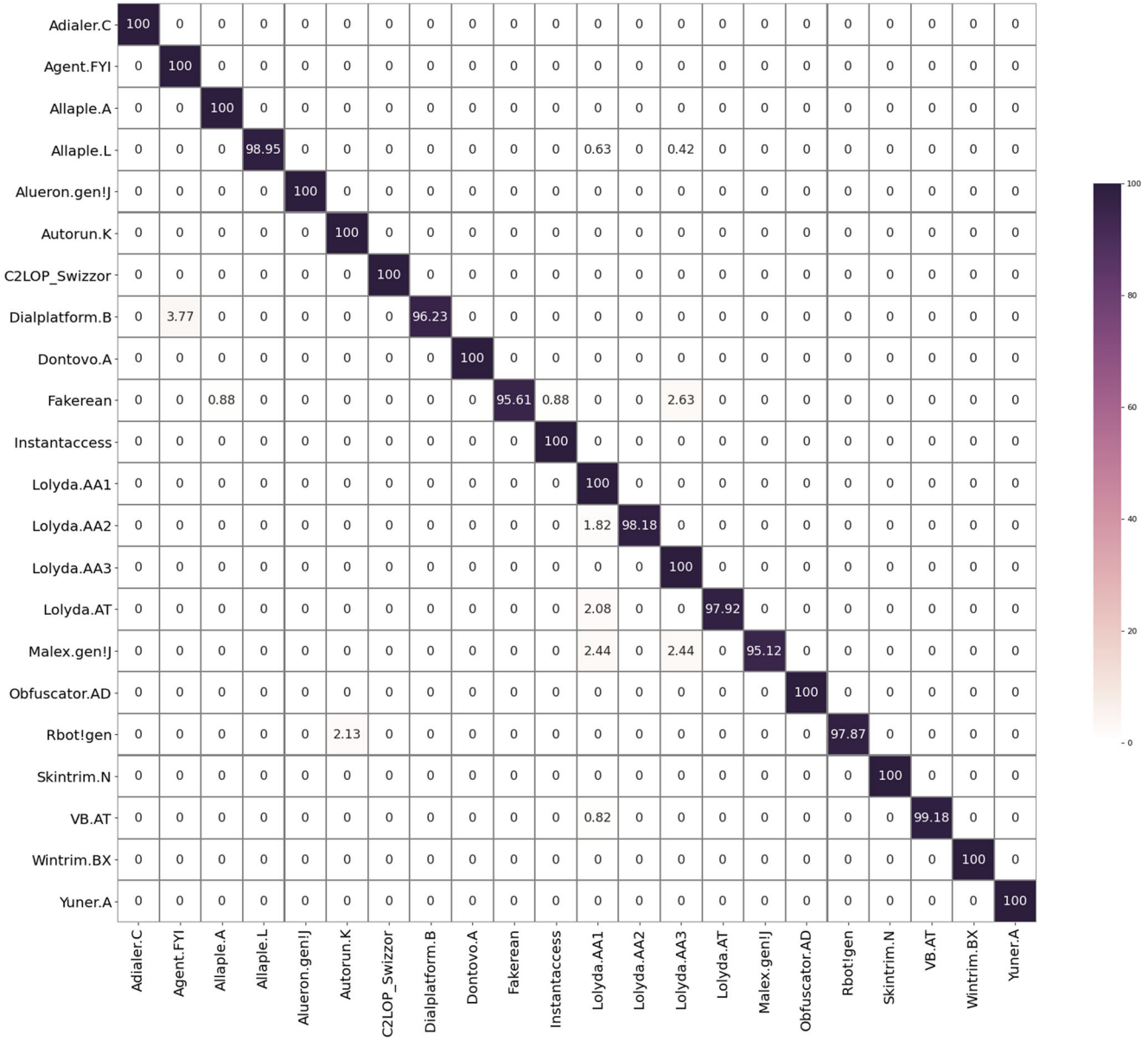


Fig. 13 Confusion matrix combining C2LOP and Swizzor families of Malimg dataset

both in terms of values of evaluation metrics and execution time.

5.6 Performance evaluation of DenseNet model on evasion attack: adversarial attacks

To evaluate the robustness of classification models, we performed adversarial attacks. In general, attacks can be categorized as poisoning attacks and evasion attacks. In the former, small perturbations are added to the training samples which forces the classifier to learn a complex hypothesis function. Consequently, the model creates overlapping decision surfaces; eventually, the model wrongly classifies the submitted samples. While in the evasion attack, samples in the test set are modified to increase the misclassification rate. In this work, we launched an evasion attack with black-box access to classification models. In particular, the adversary has zero knowledge about the classification algorithms and parameters, but may have access to surrogate samples available in public malware repositories.

In our proposed method, all executables are transformed into images; the model is developed by training the classifier on the image textures. For fabricating attacks, we add two types of additive noise (at a noise rate of 0.01) to the samples in the test set. We created evasive samples by injecting (a) Gaussian and (b) Poisson's noise. A drop in the F1 score obtained is shown in Table 5. The misclassification

Table 5 F1 score of Densenet for Malimg and Big2015 dataset

Noise	Malimg	BIG2015
Poisson's	0.971	0.907
Gaussian	0.97	0.901

is insignificant for Malimg which is a highly imbalanced dataset, but an 8% decline in F1 is obtained for BIG2015 samples. One of the important aspects that were noticed in this dataset is the presence of structural dissimilarity among variants in families, which further increased the addition of noise, causing the classifier to mislabel such instances. In the future, we would like to evaluate the performance of DenseNet by augmenting the training set with adversarial examples using the executable images added with additive noise.

6 Conclusion and future work

In this paper, we developed solutions to detect and classify malicious executables by utilizing the visualization approach. We used a deep convolutional neural network (VGG-3) and DenseNet to extract features from images. Our study analyzes the F1 measure of each model and the best F1 measure is obtained for 256×256 Gabor Markov images. Comprehensive experiments on two public benchmark dataset (Malimg and BIG2015) using DenseNet with Gabor Markov images resulted in 99.94% and 98.88% F1-measure in case of malware detection problem, while F1-measures of 99.37% and 98.88% were obtained for malware classification (family classification). Besides, comparing with the state-of-the-art, we observed improved performance in terms of both detection, classification, and execution time.

In the future, our system can be extended to evaluate the strength of the classification model against the adversarial attacks. This investigation will be conducted by injecting bytes from benign executables in the malware files, which would change the statistical distribution of the malware samples. In order to develop countermeasures on attacks, we will propose a model in which the classification algorithm will be trained with $N + 1$ classes, where N is the original malicious class and one adversarial class. In addition, we would like to investigate generating a classification model using one-shot training techniques, this would identify evolving malware families having fewer malicious variants.

References

1. BBC N Cyber-attack: Europol says it was unprecedented in scale. http://www.bbc.com/news/world-europe-39907965
2. https://blog.logsign.com/the-biggest-cyber-attacks-in-2019/ (accessed Jan 1, 2021)
3. Internet security threat report 2021, https://www.sophos.com/en-us/medialibrary/pdfs/technical-papers/sophos-2021-threat-report.pdf
4. https://crowdresearchpartners.com/portfolio/byod-mobile-security-report/(accessed Jan.1, 2021)
5. AV-Test Threat Report: https://www.av-test.org/de/statistiken/malware/ (accessed Jan 1, 2020)
6. Kang H, Jang J, Mohaisen A, Kim HK (2015) Detecting and Classifying Android Malware Using Static Analysis along with Creator Information. Int J Distrib Sens Netw **11**(6)
7. Han W, Xue J, Wang Y, Huang L, Kong Z, Mao L (2019) MalDAE: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. Comput Secur 83:208–233
8. Pascanu R, Stokes JW, Sanossian H, Marinescu M, Thomas A (2015) Malware classification with recurrent networks. In: Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP), pp 1916–1920.
9. Vinayakumar R, Alazab M, Soman KP, Poornachandran P, Venkatraman S (2019) Robust Intelligent Malware Detection Using Deep Learning. IEEE Access 7:46717–46738
10. Nataraj L, Karthikeyan S, Jacob G, Manjunath BS (2011) Malware Images: Visualization and Automatic Classification. In: Proc. 8th International Symposium, Visualization for Cyber Security, VizSec
11. Fu J, Xue J, Wang Y, Liu Z, Shan C (2018) Malware Visualization for Fine-Grained Classification. IEEE Access 6:14510–14523
12. Zhong W, Gu F (2019) A Multi-Level deep learning system for malware detection. Expert Syst Appl 133:151–162
13. Hazra A, Choudhary P, Sheetal Singh M (2021) Recent Advances in Deep Learning Techniques and Its Applications: An Overview. In: Rizvanov A. A, Singh B. K, Ganasala P (eds) Advances in Biomedical Engineering and Technology, Lecture Notes in Bioengineering. Springer, Singapore, pp 103–122
14. Gibert D, Mateu C, Planes J, Vicens R (2018) Classification of malware by using structural entropy on convolutional neural networks the thirtieth. AAAI conference on innovative applications of artificial intelligence (IAAI-18)
15. Agarap AF, Pepito FJH (2017) Towards building an intelligent anti-malware system: A deep learning approach using support vector machine (SVM) for malware classification, [online] Available: 1801.00318
16. Le Q, Boydell O, Namee BM, Scanlon M (2018) Deep learning at the shallow end: Malware classification for non-domain experts. Digit Invest 26(Supplement):S118–S126
17. Jain A, Phanishayee A, Mars J, Tang L, Pekhimenko G (2018) Gist: Efficient Data Encoding for Deep Neural Network Training. ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, pp 776–789
18. Zhang J, Qin Z, Yin H, Ou L, Zhang K (2019) A feature-hybrid malware variants detection using CNN based opcode embedding and BPNN based API embedding. Comput Secur 84:376–392
19. Kim T, Kang B, Rho M, Sezer S, Im EG (2019) A multimodal deep learning method for android malware detection using various features. IEEE Trans Inf Forensic Secur 14(3):773–788
20. Nezhadkamali M, Soltani S, Seno SAH (2017) Android malware detection based on overlapping of static features. 7th International Conference on Computer and Knowledge Engineering (ICCKE)
21. Vasan D, Alazab M, Wassan S, Naeem H, Safaei B, Zheng Q (2020) IMCFN: Image-Based Malware Classification using Fine-tuned Convolutional Neural Network Architecture. Comput Netw 171

22. Shezan FH, Afroze SF, Iqbal A (2017) Vulnerability detectio n in recent Android apps: An empirical study. 2017 International Conference on Networking, Systems and Security (NSysS)
23. Xue D, Li J, Lv T, Wu W, Wang J (2019) Malware Classification Using Probability Scoring and Machine Learning. IEEE Access 7:91641–91656
24. Yoo S, Kim S, Kim S, Kang BB (2020) AI-hydra: Advanced hybrid approach using random forest and deep learning for malware classification. Inf Sci 546:420–435
25. Zhao Y-l, Qian Q (2018) Android Malware Identification Through Visual Exploration of Disassembly Files. Int J Netw Secur 20(6):1061–1073
26. Han K, Kang B, Im EG (2014) Malware analysis using visualized image matrices. Sci World J:1–15
27. Yuan B, Wang J, Liu D, Guo W, Wua P, Bao X (2020) Byte-level Malware Classification Based on Markov Images and Deep Learning. Comput Secur 92
28. Kalash M, Rochan M, Mohammed N, Bruce NDB, Wang Y, Iqbal F (2018) Malware Classification with Deep Convolutional Neural Networks. 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, pp 1–5
29. Roseline SA, Geetha S, Kadry S, Nam Y (2020) Intelligent Vision-Based Malware Detection and Classification Using Deep Random Forest Paradigm. IEEE Access 8:206303–206324
30. Gibert D, Mateu C, Planes J, Vicens R (2019) Using convolutional neural networks for classification of malware represented as images. J Comput Virol Hacking Techn 15(1):15–28
31. Khan RU, Zhang X, Kumar R (2019) Analysis of ResNet and GoogleNet models for malware detection. J Comput Virol Hacking Techn 15(1):29–37
32. Naeem H, Ullah F, Naeem MR, Khalid S, Vasan D, Jabbar S, Saeed S (2020) Malware detection in industrial internet of things based on hybrid image visualization and deep learning model. Ad Hoc Netw 10:102154
33. Tuncer T, Ertam F, Dogan S (2021) Automated malware identification method using image descriptors and singular value decomposition. Multimed Tools Appl:1–20
34. Çayır A, Unal U, Dağ H (2021) Random CapsNet forest model for imbalanced malware type classification task. Comput Secur 102:102133
35. Sun G, Qian Q (2021) Deep Learning and Visualization for Identifying Malware Families. IEEE Trans Depend Sec Comput 18(1):283–295
36. Feng R, Chen S, Xie X, Meng G, Lin S-W, Liu Y (2021) A Performance-Sensitive Malware Detection System Using Deep Learning on Mobile Devices. IEEE Trans Inf Forensic Secur 16:1563–1578
37. Cui Z, Xue F, Cai X, Cao Y, Wang G-g, Chen J (2018) Detection of Malicious Code Variants Based on Deep Learning. IEEE Trans Ind Inform 14(7)
38. Portable freeware collection, https://www.portablefreeware.com/ (accessed Feb 1, 2020)
39. Softonic, https://en.softonic.com/windows (accessed March 1, 2020)
40. Sourceforge https://sourceforge.net/ (accessed March 1, 2020)
41. DriverPack solution, https://drp.su/en (accessed February 18, 2020)
42. Ronen R, Radu M, Feuerstein C, Yom-Tov E, Ahmadi M (2018) Microsoft Malware Classification Challenge, arXiv: 1802.10135
43. Yajamanam S, Selvin VRS (2018) Fabio di troia and Mark Stamp. Deep Learning versus Gist Descriptors for Image-based Malware Classification. 4th International Conference on Information Systems Security and Privacy, pp 553–561
44. Huang G, Liu Z, van der ML, Weinberger KQ (2017) Densely connected convolutional networks 2017. IEEE conference on computer vision and pattern recognition (CVPR)
45. Hussain M, Jordan J, Bird JJ, Faria DR (2018) A Study on CNN Transfer Learning for Image Classification. Proceedings of 18th Annual UK Workshop on Computational Intelligence Nottingham
46. Rezende E, Ruppert G, Carvalho T, Theophilo A, Ramos F, de Geus P (2018) Malicious software classification using VGG16 deep neural network's bottleneck features. In: Information Technology-New Generations. Springer, Cham, pp 51–59
47. Hazra A (2021) A comprehensive survey on chest diseases analysis: technique, challenges and future research directions. International Journal of Multimedia Information Retrieval
48. Choudhary P, Hazra A (2019) Chest disease radiography in twofold: using convolutional neural networks and transfer learning. Evolving Systems
49. C2LOP: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/C2Lop.gen!M(Accessed on 2-December,2020)