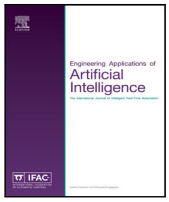
Engineering Applications of Artificial Intelligence 122 (2023) 106030

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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



A novel deep learning-based approach for malware detection

Kamran Shaukat [a,b,*], Suhuai Luo [a], Vijay Varadharajan [c]

[a] School of Information and Physical Sciences, The University of Newcastle, Australia
[b] Department of Data Science, University of the Punjab, Lahore 54890, Pakistan
[c] Advanced Cyber Security Engineering Research Centre (ACSRC), The University of Newcastle, Australia

ARTICLE INFO

Keywords:
Malware detection
Cybersecurity
Machine learning
Deep learning
Transfer learning
Ensembling
Support vector machine
Modelling
Malware
Image-based malware detection
Convolutional neural network
Classification
Cyberattack

ABSTRACT

Malware detection approaches can be classified into two classes, including static analysis and dynamic analysis. Conventional approaches of the two classes have their respective advantages and disadvantages. For example, static analysis is faster but cannot detect the malware variants generated through code obfuscation, whereas dynamic analysis can effectively detect variants generated through code obfuscation but is slower and requires intensive resources. This paper proposes a novel deep learning-based approach for malware detection. It delivers better performance than conventional approaches by combining static and dynamic analysis advantages. First, it visualises a portable executable (PE) file as a coloured image. Second, it extracts deep features from the colour image using fine-tuned deep learning model. Third, it detects malware based on the deep features using support vector machines (SVM). The proposed method combines deep learning with machine learning and eliminates the need for intensive feature engineering tasks and domain knowledge. The proposed approach is scalable, cost-effective, and efficient. The detection effectiveness of the proposed method is validated through 12 machine learning models and 15 deep learning models. The generalisability of the proposed framework is validated on various benchmark datasets. The proposed approach outperformed with an accuracy of 99.06% on the Malimg dataset. The Wilcoxon signed-rank test is used to show the statistical significance of the proposed framework. The detailed experimental results demonstrate the superiority of the proposed method over the other state-of-the-art approaches, with an average increase in accuracy of 16.56%. Finally, to tackle the problems of imbalanced data and the shortage of publicly available datasets for malware detection, various data augmentation techniques are proposed, which lead to improved performance. It is evident from the results that the proposed framework can be useful to the defence industry, which will be helpful in devising more efficient malware detection solutions.

1. Introduction

Internet is a major source of information sharing among distinct nodes. It consists of millions of computers, networks, and devices. Consequently, Internet has become the target of cybercriminals. Malware, spam, and phishing are examples of such cyberattacks (Shaukat et al., 2020b). Malware is considered to be one of the major security threats to cyberspace. Malware is a combination of 'mal' from 'malicious' and 'ware' from 'software'. It is a code snippet covertly inserted into a computer system or a network with malicious intent to disrupt the normal flow of activities. Based on its functionality, sometimes malware is classified into viruses, worms and Trojan horses (Guo et al., 2016).

Cybercriminals use various obfuscation methods to generate malware variants. Exclusive-OR, base64 encoding, ROT13, dead code insertion, instruction substitution, and sub-routine reordering are common obfuscation methods. These methods add malicious adversaries into binary and textual data that malware detectors find difficult to interpret and detect (Shaukat et al., 2022). The new malware and unwanted applications are reported to be over 450,000 daily and over 100 million yearly. The malicious pattern can reside within a data file or software application, to name a few. These software applications may belong to different platforms, such as Windows, Linux, and Android.

Malware analysis techniques are usually categorised into two major groups: static and dynamic. In static analysis, an application is observed for malicious patterns without execution. These data files or applications are decrypted and disassembled into feature vectors. Feature vectors characterise the essential features and format information of the file potentially containing the malicious pattern. Anti-malware solutions detect malicious patterns by analysing these feature vectors. Conventional malware detectors use a signature-based detection method. These detectors are stored with a vast database of malware signatures (malicious code patterns). They decode the suspicious file and match its extracted static features with the stored malware signatures. Malware variants can easily undermine conventional anti-malware solutions.

* Corresponding author.
E-mail address: kamran.shaukat@uon.edu.au (K. Shaukat).

https://doi.org/10.1016/j.engappai.2023.106030
Received 6 November 2022; Received in revised form 7 February 2023; Accepted 19 February 2023
Available online 9 March 2023

Static analysis is vulnerable to adversarial attacks and obfuscation techniques. Dynamic analysis is carried out by running the application in a controlled virtual environment to observe the dynamic behaviour and potential malicious pattern (Arora et al., 2014). Function call monitoring, dynamic visual analysis, and instructions traces are common techniques used in dynamic analysis. Typically, dynamic analysis is implemented using a sandbox. Researchers have also combined static and dynamic analysis to detect malware (De Paola et al., 2018).

Challenges: The exponential growth of malware variants is another challenge for conventional anti-malware solutions in addition to the challenges mentioned above. These solutions are incapable of detecting these polymorphic and rapidly growing malware variants. The static analysis is faster and provides insights into the structural properties of the application; thus can detect the malware variant within a complete area of coverage. However, a detailed analysis is impossible as most variants employ detection avoidance techniques that are generated using code obfuscation. On the other hand, dynamic analysis effectively detects variants generated through code obfuscation. Nevertheless, it is slower, requires intensive resources (virtual environment, i.e., Sandbox) and is highly time-consuming. It requires human effort to analyse and interpret detailed reports. There is where learning-based artificial intelligence-based techniques came into practice. Machine learning is a sub-branch of artificial intelligence (AI) that learns from the existing attack patterns and prevents similar attacks in the future. Cyber defenders have also used machine learning-based methods to detect suspicious characteristics in evolving malware variants (Li et al., 2021). However, it is still challenging for machine learning-based methods to detect polymorphic and new malware variants. Nataraj et al. (2011) proposed a method to convert a malware-packed binary into a greyscale image to classify different malware families. Their detection approach considered the visible features to address code obfuscation. Image-based malware conversion can help to overcome the problem of detection avoidance techniques in certain malware variants. However, extraction of complex texture features, i.e., Super-Resolution Radial Fluctuations (SRRF) and Linde-Buzo-Gray (LBG), slowed the detection processing.

Consequently, the representation of various variants that can be represented in colour images is overlooked. Colour images are represented using red, green, and blue (RGB) colour format. The proposed approach overcomes these challenges by providing an efficient mechanism for converting a PE file into a coloured image. Thus, the proposed method reduces the challenges given by static and dynamic methods.

Learning-based detectors come with their challenges. They need to be trained with a sufficient and equal number of instances from each class (benign and malicious). Due to privacy and security issues, most datasets that contain the latest attacks are private. The publicly available datasets are laboriously anonymised and suffer from various issues. The problem of imbalanced data is a major concern for the learning-based detector as few malicious samples are present to train the detector properly. The skewed distribution of the training examples makes standard learning-based malware detectors biased towards the majority class. The proposed approach targeted the imbalanced dataset and enhanced the detection effectiveness by applying various data augmentation techniques.

Machine learning models are not explicitly designed for malware detection tasks and need domain knowledge. However, deep transfer learning pre-trained models are trained with more than 10 million images. These cutting-edge pre-trained models can be directly used as detectors (end-to-end classification) and feature extractors for malware image classification. Thus, combining both machine learning (ML) and deep learning (DL) can help to overcome the intensive feature engineering and domain knowledge tasks. Hence, we have combined recent developments in image processing using transfer learning with state-of-the-art machine learning to improve detection accuracy. This paper proposes a novel hybrid malware detector using a deep learning model as a feature extractor and a machine learning model as a detector. The proposed approach targeted all the aforementioned challenges. The performance of the proposed approach is validated with 15 deep learning and 12 machine learning models. We demonstrate that our proposed approach outperforms the state-of-the-art conventional and learning-based solutions.

Our contributions are as follows:

- We propose a novel hybrid framework combining deep transfer learning and machine learning for malware detection. First, deep transfer learning is used to extract all the deep features from the last fully connected layer of the deep learning model, and then machine learning models are used as the final detector, which fully utilises the inherent connections between input and output. The proposed approach eliminates the need for knowledge from domain experts for reverse engineering tasks.
- The proposed framework consists of two steps. In the first step, an image-based PE dataset is generated by transforming malicious and benign Windows executables into RGB-coloured images. We address the imbalanced data problem in this step using data augmentation. The images are normalised at this stage. The performance of 15 deep transfer learning models is evaluated on three datasets for image-based malware detection. We show that data augmentation enhances the detector's performance.
- The second step involves a novel combination of deep learning and machine learning models for malware detection. An in-depth analysis of 15 deep learning models as feature extractors and 12 end-to-end machine learning models as malware detectors is presented.
- We demonstrate that our proposed approach is scalable, cost-effective, and efficient. The effectiveness of the proposed framework is validated on a smaller dataset. We analyse the effectiveness of various models, first considering only a single feature and then using all features for malware classification. Our results demonstrate that the proposed framework performs better than other state-of-the-art techniques.

The rest of the paper is structured as follows. Section 2 provides a literature review of the relevant related works. Section 3 describes the proposed framework. Results and discussion are given in Section 4. Section 5 concludes the paper and outlines future research. We list the acronyms used in this paper in Table 1 for convenient referencing.

2. Related work

Most anti-malware solutions are based on either conventional or learning-based solutions. The conventional anti-malware solution includes feature extraction, analysis mode (static and dynamic), and statistical similarity measures. Learning-based anti-malware solutions use machine learning, deep learning, and visualisation techniques. Subsequent sections describe related works in each of these methods.

2.1. Conventional anti-malware approaches

Ishai et al. in Rosenberg et al. (2018) proposed a method to classify advanced persistent threats (APTs). APTs are a collection of highly evolving, dynamic and sophisticated attacks developed by nation states. The authors used dynamic analysis using a sandbox to extract features of APTs. A deep neural network (DNN) was further used for detection. Yu Jiang et al. in Jiang et al. (2020) proposed a method based on static Android malware analysis. The authors have extracted the opcode sequence of malware and provided a detection mechanism. However, their approach did not handle the packed applications. Packed applications have encrypted applications that can be restored only when the application is executed. Their proposed system also ignored Android applications that use dynamic loading technology (DLT). The Android applications that use DLT are not detectable using static analysis. Zhangjie Fu et al. proposed a long short-term memory for Android

Table 1
List of abbreviations and acronyms.

Abbreviation	Description	Abbreviation	Description
AI	Artificial Intelligence	LDA	Linear Discriminator Analysis
ANN	Artificial Neural Network	LR	Logistic Regression
APTs	Advanced Persistent Threats	LSTM	Long Short-Term Memory
APK	Android Application Package	ML	Machine Learning
API	Application Programming Interface	MLP	Multi-Layer Perceptron
CNN	Convolutional Neural Network	NB	Naïve Bayes
DT	Decision Tree	NN	Neural Network
DL	Deep Learning	PE	Portable Executable
DNN	Deep Neural Network	PCA	Principal Component Analysis
DLL	Dynamic-Link Libraries	RF	Random Forest
FNR	False Negative Rate	RNN	Recurrent Neural Networks
FPR	False Positive Rate	SVM	Support Vector Machine
HistGradientBoosting	Histogram-Based Gradient Boosting	TL	Transfer Learning
IoT	Internet of Things	TNR	True Negative Rate
IDS	Intrusion Detection System	TPR	True Positive Rate
k-NN	K Nearest Neighbour	XGBoost	Extreme Gradient Boosting

malware detection. They have extracted the features from 3090 benign and 3090 malicious Android application package (APK) files (Fu et al., 2021). Their experiment results outperformed traditional support vector machine (SVM), random forest (RF), decision tree (DT), and K-nearest neighbour (KNN) models.

Schultz et al. in Schultz et al. (2000) proposed a data mining-based method using static analysis. The authors extracted the byte, string sequences, and portable executable head features from malicious binaries. Instead of extracting the byte n-gram sequence as in Schultz et al. (2000), Shabtai et al. extracted the opcode sequence to identify the malicious pattern (Shabtai et al., 2012). Kolter et al. in Kolter and Maloof (2006) also performed static analysis and extracted the n-grams of byte codes from a set of benign and malicious Windows executables. The authors extracted the most relevant n-grams from a set of 255 million distinct n-grams. Naïve Bayes, SVM, and decision tree have been used for malware detection. However, their proposed method failed to correctly classify malicious codes with obfuscation and encryption. Further, all these approaches need a domain expert and feature engineering tools for malware detection.

Researchers have also analysed the API system call sequences for detecting malicious behaviour using dynamic analysis. Searles et al. in Searles et al. (2017) performed dynamic analysis to extract the dependency relationships among various system calls. They constructed a similarity matrix to feed the SVM for malware detection based on the generated dependency graph. Cesare et al. in Cesare et al. (2013) extracted the control flow graphs using de-compilers. Further, the control flow graphs were used to identify different malware variants. Malware detection was also performed through static visual analysis. Authors either performed static or dynamic analysis to identify a specific behaviour or extract specific features from the malware binaries. The monitored behaviour or specific features are transformed into an image. Afterwards, various statistical-based similarity methods are used on crafted image datasets for final malware detection. Gianni et al. in D'Angelo et al. (2020) performed dynamic analysis on Android-based apps. The authors converted the API call sequences into API images during execution. An autoencoder is then used for detection. They show that this approach outperforms the decision tree, naïve Bayesian, with an accuracy of 94%.

Syed et al. in Shaid and Maarof (2014) performed dynamic analysis to classify various malware variants. The authors executed the malware apps in a virtual environment to identify the API call sequences. The extracted API call sequences were then transformed into a coloured image. The authors applied various statistical methods to find similarities among different malware images. KyoungSoo et al. used opcode instructions to generate coloured images. The image matrices are generated from these images. Specific areas of images are used to identify similarities among these matrices (Han et al., 2013). Most researchers extracted a specific feature during dynamic analysis for further malware detection; for example, the work in Anderson et al. (2011) collected instruction traces, the work in Imran et al. (2015) observed system calls, and the work in Fujino et al. (2015) monitored API calls. Though dynamic analysis helps to identify malicious behaviour, it is a time-consuming and resource-intensive task.

The hybrid-based approach combines both static and dynamic analysis. Alejandro et al. in Martín et al. (2019) proposed a framework that can perform static and dynamic analyses. The authors performed the experiments on Android applications. Huda et al. in Huda et al. (2016) proposed a hybrid analysis approach that obtained the cluster information using term-frequency features. The authors used an SVM for malware detection.

2.2. Learning-based anti-malware approaches

Many research works have leveraged machine learning and deep learning advancements for malware detection. Many works have converted the malware binaries into images and used various machine learning models to classify and identify different malware families. This section describes the state-of-the-art malware detection approaches using visualisation-based and learning-based techniques.

Niket et al. in Bhodia et al. (2019) used transfer learning for malware detection. The authors compared traditional K-NN to ResNet34 on the Malicia dataset. The authors reported that deep learning ResNet34 outperformed traditional K-NN. However, the authors did not compare it with other state-of-the-art deep learning architectures. They have used 9895 malware instances and only 704 benign samples to train their model. They did not tackle the imbalanced data problem. Most of the researchers only used one deep learning model for malware detection by neglecting the latest deep learning architectures; e.g., Li Chen in Chen (2018) used Inception-V1, Wai Weng et al. in Lo et al. (2019) used Xception, Prima et al. in Prima and Bouhorma (2020) used VGG16, Daniel et al. in Nahmias et al. (2020) used VGG-19, and Yuntao et al. in Zhao et al. (2020) used Faster Region-Convolutional Neural Networks (RCNN) only. MalConv was initially proposed by Raff et al. (2018) for malware and benign classification. Kadri et al. in Kadri et al. (2019) used MalConv to classify different classes of malware. Kadri et al. claim that MalConv gives similar performance for classifying among various classes of malware. In Kumar (2021), authors used the Malimg dataset for malware classification by slightly modifying the last layer of ResNet50. Their proposed model is better than the standard ResNet50. However, the authors have not provided a rationale for using ResNet50 only. Furthermore, there was no analysis provided for other state-of-the-art deep learning architectures. Xianwei Gao et al. in Gao et al. (2020) extracted byte features from malware and applied recurrent neural network (RNN) for malware detection. The authors also validated the performance of their approach using various machine learning models and XGBoost model comes out the best. However, their approach has the overhead of static analysis and feature selection.

Edmar et al. in Rezende et al. (2017) used ResNet50 to classify malware families. The authors used the Malimg dataset and compared their approach with Nataraj et al. (2011). The authors did not provide a comparison with state-of-the-art deep learning architectures. The authors did not address the imbalanced number of instances within each malware family. Walid El-Shafai et al. in El-Shafai et al. (2021) used VGG16 and InceptionV3 to classify different malware families. They also have not handled the imbalanced data problem while training their models. Mazhar Javed Awan et al. used the Spatial Attention mechanism with VGG19 to classify various malware variants using the Malimg dataset (Awan et al., 2021). Danish Vasan et al. in Vasan et al. (2020b) used VGG16 and ResNet50 pre-trained models to extract features from the Malimg malware dataset. Then they ensembled the output of both models to do the final classification among different malware families. Again, they did not consider the problem of the imbalanced number of instances among different types of malware. Trojan class has only 80 instances, whereas Worm has 2949 instances. Secondly, in real-world scenarios, the essential work is to classify between malware and non-malware files rather than classifying among different families of malware. The authors did not validate their proposed approach on real-world task of classifying a malware and benign software. Pratikkumar et al. in Prajapati and Stamp (2021) used VGG19 and ResNet152 to classify among 20 families of malware. The malware dataset consisted of different families of malware from the Malicia dataset and Kim (2018). VGG19 gave the best accuracy of 92.16%.

2.2.1. Data resampling and augmentation

Researchers have been using various data resampling techniques to balance their data. Data resampling techniques remove the skewed distribution towards a specific data class and balance the data in the training process. Niccolò et al. in Marastoni et al. (2021) applied data augmentation techniques to increase the dataset size for model training. The authors applied long short-term memory (LSTM) and convolutional neural network (CNN) models for malware detection. The authors claim that data augmentation increases detection accuracy. In addition, the authors claim that the models trained on a malware dataset can be used to classify the malware samples from another dataset.

Zhihua Cui et al. in Cui et al. (2019) argued that the Malimg dataset is highly imbalanced. The authors used under-sampling techniques (removing instances from a majority class) to balance the data. Bouchaib Prima et al. in Bouchaib and Bouhorma (2021) used the synthetic minority oversampling technique (SMOTE) to balance the data. The authors then applied VGG16 for classification among different classes of malware.

2.2.2. Application domains: IoT and network traffic

Researchers also use deep learning models for malware detection on other domains, such as the Internet of Things (IoT) and Android malware. Karanja et al. used VGG19 for malware detection for IoT malware. The authors have experimented on a set of 258 IoT malware samples. The dataset used for the study was minimal, and only VGG19 architecture was used for experiments. Wei Wang et al. in Wang et al. (2019) used deep autoencoder and CNN to classify the Android malware.

Visualisation-based techniques are also being applied in other areas of network analysis. Candong Rong et al. in Rong et al. (2020) converted the network traffic from the environment into RGB images. The authors compared their malicious traffic detection approach with VGG19 and ResNet50. The authors claimed that their proposed approach outperformed other deep learning models in classifying the different variants of malicious traffic. Authors in Chandio et al. (2022) and Roy et al. (2022) provided an improved deep learning-based model for feature extraction. The authors further used these models for object detection in an agricultural field. Authors in Jiang et al. (2022) provided an efficient deep learning-based method using graph data for any classification task. Table 2 presents a compact comparison of the related works. Extensive experiments are performed with 27 (Used Model Count) in this paper.

3. Methodology

This section describes our proposed malware detection framework and discusses applied methods. The proposed framework is composed of two steps. In the first step, an image-based PE dataset is generated by transforming all the malign and benign PEs into images. This step also targets the imbalanced data problem. The details are given in Section 3.1. Once the image-based PE dataset is pre-processed, the second step is initiated. The second step comprises deep learning and machine learning models for malware detection. The details of the second step are given in Section 3.2.

3.1. Dataset and imbalanced data problem

This section is divided into three sub-sections. First section, 3.1.1, describes the details of the datasets used in this paper. We have translated vision-based image classification methods into malware classification methods. Second section, 3.1.2, explains the conversion process of malware and benign PE files into images. Here we also present the API call extraction methods using static analysis and converting those API feature vectors into images for further detection to feature vectors into images for further malware detection. Third section, 3.1.3, highlights the imbalanced data problem and its proposed solution.

3.1.1. Dataset

This section briefly summarises the different datasets used in this study. Our experiments are performed on Windows Portable Executable (PE) files. The PE format is a file format in the Windows operating system (OS) used for executables, object code, and dynamic-link libraries (DLL). The malicious PEs are collected from three datasets, including the Microsoft malware dataset (Microsoft, 2015), Malimg (Nataraj et al., 2011), and VirusShare (VirusShare, 2022). Multiple online resources, including softpedia.com, download.com, and Internet download sites (Al-Dujaili et al., 2018), have been used to collect benign PEs. The malicious PEs corpus collected from the Malimg dataset is used to validate the performance of the proposed framework. The dataset ration for training and validation is used similarly to state-of-the-art methods.

Additionally, the PEs corpus collected from the VirusShare dataset is used to evaluate the generalisation effect of our proposed malware detector. During the evaluation, the notorious Conficker worm was also added to the testing dataset to validate the detection effectiveness of our proposed malware detector against zero-day attacks. Microsoft malware classification challenge was first held in 2015. The dataset contains malware from nine families of malware. The Malimg dataset contains 9435 real-life malware samples of 25 different families (Nataraj et al., 2011). This dataset is widely used in the literature for visualisation-based malware classification tasks. VirusShare is a large collection of real-life malicious executables. The corpus of malicious samples is updated continuously, containing 47,132,110 malware samples (VirusShare, 2022).

3.1.2. Transformation of malware and benign PEs into images

To transform malware detection methods into a vision-based image detection problem, the benign and malicious PEs are converted into coloured images (red, green, blue (RGB)). PEs can be visualised into images using two methods. A PE file is translated into a raw byte binary stream. Image is constructed through this raw byte binary steam. Alternatively, the PE file can be disassembled into raw binary to obtain an .asm file. The .asm file is then converted into an image. A .asm file loses the structural pattern of the PE file, hence the loss of essential features (Naeem et al., 2019). This study converted the PEs into images using raw byte binaries to avoid any loss of essential features. Fig. 1 illustrates the conversion process.

A raw byte binary stream of a PE file is divided into substrings of 8 bits in length. Each substring of 8 bits is preserved as a pixel

Table 2
A comparative analysis of the literature.

Sr#	Ref#	Year	Used model count	Models	Feature extraction	Need for domain expert	Transfer learning	Technique static/dynamic/ Visual	Handled data imbalance	ML	Dataset	Deep features extracted	Experimented on smaller dataset	One-vs-All features comparison
1	Nataraj et al. (2011)	2011	1	KNN	No	No	No	Visual	No	Yes	Malimg	No	No	No
2	Rosenberg et al. (2018)	2018	1	DNN	Yes	Yes	No	Dynamic	No	No	APTs	No	No	No
3	Jiang et al. (2020)	2020	1	Resnet50	Yes	Yes	Yes	Static and visual	No	No	Drebin	No	No	No
4	Fu et al. (2021)	2021	1	LSTM	Yes	Yes	Yes	Static	No	No	VirusShare	No	No	No
5	Shabtai et al. (2012)	2012	5	ANN, LR, RF, DT, NB,	Yes	Yes	No	Static	No	Yes	VXHeaven	No	No	No
6	Kolter and Maloof (2006)	2006	4	NB, DT, SVM, Boosting	Yes	Yes	No	Static	No	Yes	VXHeaven	No	No	No
7	Searles et al. (2017)	2017	1	SVM	Yes	Yes	No	Dynamic	No	Yes	Custom	No	No	No
8	D'Angelo et al. (2020)	2020	1	Autoencoder	Yes	Yes	No	Dynamic	No	No	Custom	No	No	No
9	Chen (2018)	2018	1	InceptionV1	No	No	Yes	Visual	No	No	Microsoft	No	No	No
10	Lo et al. (2019)	2018	1	Xception	No	No	Yes	Visual	No	No	Malimg, Microsoft	No	No	No
11	Kumar (2021)	2021	1	ResNet50	No	No	Yes	Visual	No	No	Malimg	No	No	No
12	Hemalatha et al. (2021)	2021	1	DenseNet	No	No	Yes	Visual	Yes	No	Malimg	No	No	No
13	Gibert et al. (2019)	2019	1	CNN	No	No	No	Visual	No	No	Malimg	No	No	No
14	Vasan et al. (2020b)	2020	2	VGG16, ResNet50	No	No	Yes	Visual	No	No	Malimg	No	No	No
15	Roseline et al. (2019)	2019	4	RF, Xgboost, ETC, LR	Yes	No	No	Visual	No	Yes	Malimg	No	No	No
16	Ben Abdel Ouahab et al. (2019)	2019	1	KNN	Yes	Yes	No	Visual	No	Yes	Malimg	No	No	No
17	Vinayakumar et al. (2019)	2019	5	CNN, DT, NB, LR, KNN	Yes	Yes	No	Dynamic and Visual	No	Yes	Malimg	No	No	No
18	Rong et al. (2020)	2020	1	ResNet50	No	No	Yes	Visual	No	No	MCFP	No	No	No
19	Rezende et al. (2017)	2017	1	ResNet50	No	No	Yes	Visual	No	No	Malimg	No	No	No
20	Bansal et al. (2021)	2021	4	VGG19, NB, DT, RF	Yes	No	Yes	Visual and Static	No	Yes	Caltech-101	No	No	No
21	**Our paper**	**2022**	**27**	**–**	**No**	**No**	**Yes**	**Static, Visual**	**Yes**	**Yes**	**VirusShare, VXHeaven, Malimg**	**Yes**	**Yes**	**Yes**

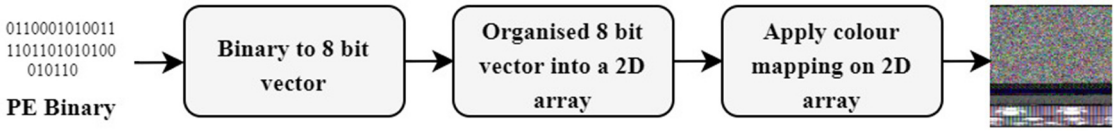


Fig. 1. An illustration of malware transformation into an image.

in an image. Afterwards, the substring of length 8 bits is converted into an unsigned integer ranging between 0–255. For example, the PE binary given in Fig. 1 is divided into substrings of 8 bits length, i.e., 01100010100111101101010100010110 → 01100010, 10011110, 11010101, 00010110 → 98, 158, 213, 22. A one-dimensional (1D) feature vector containing decimal values of 8 bits substrings is generated. The 1D feature vector is transformed into a two-dimensional (2D) matrix. The height of the image depends upon the contents of the PE file. However, the width of the image is fixed according to criteria based on the file size. The width of the image is determined using the file size of the PE file (Oliva and Torralba, 2001). Finally, a colour map is applied to the 2D matrix to visualise the PE binary. Most of the research in visualisation-based malware detection was performed with greyscale images. The colour images better represent various malware variants and other features than greyscale images (Vasan et al., 2020a). We have performed all our experiments on coloured images.

PE file contains information related to API calls, Opcodes, etc. We have generated an API-image dataset by extracting the API calls from each PE file. The purpose of generating this API-based image dataset is to analyse the detection effectiveness of learning-based detectors on an image dataset generated through one feature or all features. A set of experiments was performed to compare the detection effectiveness of malware detection using an image dataset generated by converting a whole PE file binary into an image or through an API-based image dataset. API-based image dataset is generated by extracting the API calls. For brevity, we have used the same malware and benign PEs dataset from Al-Dujaili et al. (2018). The Al-Dujaili et al. (2018) dataset contains 19,000 malicious and 19,000 benign PE files. LIEF (LIEF Library, 2021) library is utilised to parse each PE file. There were 22,760 unique API calls in the Al-Dujaili et al. (2018) dataset. Thus, the size of each binary feature vector is 22,761. After parsing, each PE file is mapped into its corresponding binary feature vectors. Each index of the feature vector represents the presence or absence of a unique API call. '1' on the respective index denotes the presence of an API call, '0' otherwise. Each feature vector is mapped into an API-based image dataset using the process illustrated in Fig. 1.

Thus, the whole image generation and later detection process do not require any feature engineering or domain expert. The height and width of the image are depended upon the contents and the size of the PE file, respectively. All the images are pre-processed and normalised according to the network input settings. Most of the deep learning architectures take input images of size 224 × 224. Thus, we have transformed all the images into a fixed rectangular shape of size 224 × 224. However, the images have been resized in case some deep learning-based models take different image sizes as input. We have resized the

Table 3
Augmentation techniques and respective values.

Sr#	Technique	Value	Sr#	Technique	Value
1	Rescale	1./255	6	Height shift	0.1
2	Shear range	0.2	7	Width shift	0.1
3	Vertical flip	True	8	Brightness	0.9,1.1
4	Horizontal flip	True	9	Fill mode	Nearest
5	Zoom range	0.2	10	Rotation range	0.0

images in these cases. Though some features are lost during this resizing process, texture features are preserved.

3.1.3. Data imbalance and solution

Class imbalance and unavailability of the most recent attack patterns are well-known problems in malware detection. Due to privacy reasons, data on the latest attacks are not publicly available. This makes the learning models skewed and biased towards the majority class. The malware classes in benchmark and well-known public datasets are not equally distributed. The Malimg dataset comprises 9339 malware samples from 25 families and 617 benign samples. Among different families of malware, the Allaple. A family has 2949 samples, whereas the Skintrim.N has only 80 samples. Microsoft malware dataset is composed of 10 868 malware samples. Out of 10 868 samples, the Kelihos_ver3 malware family has 2942 samples, and Simda has only 42 samples. As mentioned in Table 2, this problem has not been addressed for learning-based malware detectors. Learning-based detectors are skewed towards the majority class and give less attention to the minority classes. The detector accuracy is unreliable due to the lack of consideration of minority class samples during the model's training. We have used state-of-the-art data augmentation techniques to overcome the degradation of the detection effectiveness of learning-based detectors and avoid overfitting caused by an imbalanced dataset.

The real image data is used to augment new images by relocating the image pixels. The augmentation is performed to ensure that the structural and other important features of the images remain consistent. The augmentation not only expands the size of the data but also adds a level of variation in different samples of the dataset that allows the malware detector to generalise better on unseen data. We have used various augmentation techniques, including rotation range, width shift, height shift, zoom range, horizontal flip, vertical flip, brightness range, rescale, fill mode, and shear range. It is worth considering that a few data augmentation techniques may alter the image dimension after augmentation, e.g., rotation. Rotating a square image will preserve the image dimension. However, rotating a rectangle shape image will not preserve the dimension once rotated at 180 degrees. Considering this issue, we have already normalised all our images into square 224×224 size images. The normalisation process preserves not only the image dimension but also important features in the augmented images. Thus, the generated data is quite similar to the real data. In this study, we used the Augumentor package of Python, which has features of size-preserving rotations and size-preserving shearing. These features ensure the preservation of the original features of images in the augmented data.

We have created a pipeline of various augmentations techniques. The pipeline processes each augmentation technique sequentially. However, probability can also be assigned to each augmentation technique. Alternatively, any augmentation technique can be applied randomly, or only a subset of augmentation techniques can be applied from the pipeline. Let $A = [a_1, a_2, a_3, \ldots a_{10}]$ be the pipeline of augmentation techniques. $S_i = a_x, \ldots\ldots a_y$ be the sequence of 10 augmentation techniques given in Table 3. Each augmentation technique is assigned a specific weight W_x. Hence, the final sequence of augmentation techniques is given as $S_i = a_x w_x, \ldots\ldots a_y w_y$- where w_x is the weight assigned to each augmentation technique. We have only augmented the images for training data. The test data is not augmented. Using these augmentation techniques, we have balanced the training data of each class (malware and benign) into an equal number of instances. The augmented data will help to generalise the detector properly to each class. After these steps, an image-based PE dataset is generated and processed. Let us now consider the hybrid deep learning-based malware framework.

3.2. Proposed hybrid malware detection framework

This section explains the second part of the framework, i.e., hybrid deep learning and support vector machine (SVM) classifier for the final malware detection. The first section, 3.2.1, briefly explains the use of deep learning models and transfer learning. Then, Section 3.2.2 describes the rationale behind the proposed approach. Section 3.2.3 briefly introduces the final detector, i.e., the support vector machine and other applied machine learning models, to validate the performance of the proposed framework. The explanation of the Convolutional neural network-SVM (CNN-SVM) malware detector is provided in Section 3.2.4. Fig. 2 illustrates the pictographic explanation of the second-step of the proposed novel framework for malware detection. We name the proposed model a hybrid deep learning and machine learning-based malware detector (HDLMLMD ≡ HD(LM)^{2}D). The HD(LM)^{2}D is a framework that classifies malware and benign PEs. It consists of 3 parts: image-based PE image dataset generation (step 1 explained as Section 3.1), fine-tuned pre-trained model and transfer learning to extract features, and final detection to get an output.

Fig. 2 depicts the second step of the proposed framework. It consists of pre-processed PE Image dataset, data splitting, a pre-trained model for feature extraction, and feeding of extracted features to the final classifier. It takes the pre-processed PE images as input. The pre-processed dataset is generated as a result of step 1 (Section 3.1). The data is split into training, validation, and testing. The training and validation data are used for transfer learning and feature extraction. The testing data is used to test the detection effectiveness of the models. Any deep learning model can be used to extract the deep features from the malware image dataset. The deep features are extracted from the last fully connected layer as 1D. The extracted deep features are fed into a machine learning model, i.e., SVM, for final malware detection. The non-linear machine learning model fully utilises the inherent connections between input and output. The performance of the proposed HD(LM)^{2}D framework is compared with state-of-the-art ML and DL-based approaches. The performance of the final detector, i.e., SVM in the proposed framework, is also validated with other ML models. The following sections explain the framework in detail.

3.2.1. Deep learning models and transfer learning

Transfer learning (TL) is a machine learning method that leverages the knowledge learnt from solving a problem to solve another related problem. Transfer learning alleviates two major problems in malware detection: imbalanced datasets and inadequate computing resources. As mentioned in the previous sections, the available public datasets do not contain sufficient latest attack patterns or suffer from an imbalanced number of instances required to train a malware detector effectively by avoiding overfitting. We have addressed the problem of imbalanced data through data augmentation described earlier. Transfer learning can be formally defined in terms of domains and learning tasks. Table 4 summarises the formal notations:

Let $\mathcal{D}_s$ be a source domain, $\mathcal{T}_s$ be a source learning task, $\mathcal{D}_t$ be a target domain, $\mathcal{T}_t$ is a target learning task. The source domain $\mathcal{D}_s$ consists of a feature space $\mathcal{X}_s$ and a marginal probability distribution $P(X_s)$, where $X_s = \{x_1, x_2, \ldots, x_n\} \in \mathcal{X}_s$. If the $\mathcal{D}_s$ and $\mathcal{D}_t$ are different, then $P(X)$ and $\mathcal{X}$ would also be different, i.e., $\mathcal{X}_s \neq \mathcal{X}_t$ and $P(X_s) \neq P(X_t)$.

Given the source domain $\mathcal{D}_s$, $\mathcal{D}_s = \{\mathcal{X}_s, P(X_s)\}$, a learning task $\mathcal{T}_s$ consists of two components: a label space $\mathcal{Y}_s$, where $Y_s = \{y_1, y_2, \ldots, y_n\} \in \mathcal{Y}_s$, and mapping objective function f_s: $\mathcal{X}_s \rightarrow \mathcal{Y}_s$. The function f_s

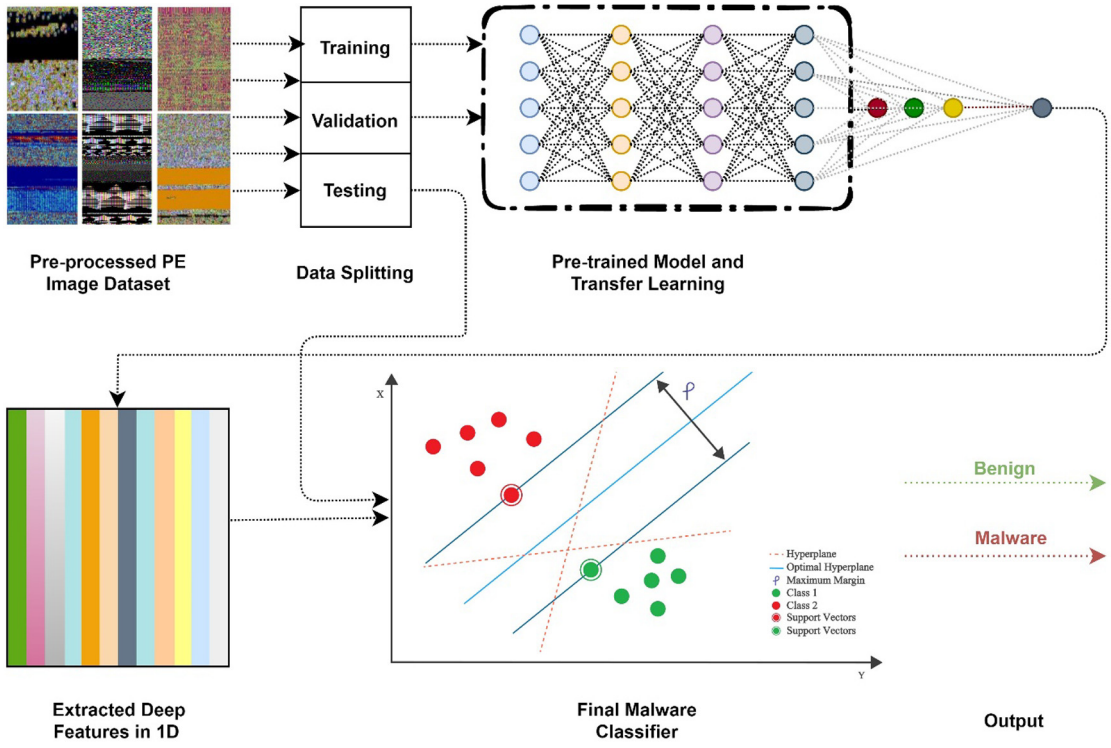


Fig. 2. The proposed malware detection framework.

Table 4
Formal notations and their descriptions.

Notations	Description	Notations	Description
$\mathcal{D}_s$	Source domain	$\mathcal{X}_s$	Source feature space
$\mathcal{T}_s$	Source learning task	$P(X_s)$	Marginal probability distribution of source feature space
$\mathcal{D}_t$	Target domain	$P(X_t)$	Marginal probability distribution of target feature space
$\mathcal{T}_t$	Target learning task	$\mathcal{Y}_s$	Label space of the source domain
f_s	Objective function	$\mathcal{Y}_t$	Label space of the target domain

(x) is used to efficiently predict the corresponding label y_{new} of a new instance x_{new}. The learning task $\mathcal{T}_s = \{\mathcal{Y}_s, f(x)\}$ is learnt through the source training data consisting of pairs $\{x_{si}, y_{si}\}$, where $x_{si} \in \mathcal{X}_s \wedge y_{si} \in \mathcal{Y}_s$.

In our case, the $\mathcal{D}_s$ is the collection of images with the ImageNet dataset, and $\mathcal{D}_t$ is the collection of malware images. $\because \mathcal{D}_s \neq \mathcal{D}_t \vee \mathcal{T}_s \neq \mathcal{T}_t$ the aim of transfer learning to utilise knowledge learned from source domain $\mathcal{D}_s$ and source learning task $\mathcal{T}_s$ for generating the target mapping objective function f_t: $\mathcal{X}_t \rightarrow \mathcal{Y}_t$ on malware domain $\mathcal{D}_t$ to improve the malware detection effectiveness. In this study, 15 pre-trained models are applied for malware detection. The architectural details of these models are given in Table 5.

The pre-trained models are trained on the ImageNet dataset. ImageNet dataset has 10+ million images containing 1000 class labels. In this study, the pre-trained models are analysed from two perspectives. First, the 15 pre-trained models are fine-tuned and used for end-to-end malware detection. We have fine-tuned all the models by adding a dropout layer, a flattened layer, and a final fully connected layer containing two classes (malware and benign) instead of a fully connected layer (containing 1000 classes). Initial weights of pre-trained models (trained on natural images) were used, and the last layers were fine-tuned for the malware detection task. An in-depth analysis of 15 malware detectors is done on three sets of datasets. Second, the 15 pre-trained fine-tuned models were used as feature extractors for the last fully connected layer. The shortcoming of the pre-trained models in accurately detecting malware images and its solution is provided in the next section. RegNetY320 has been used to extract the features. Facing the problem of vanishing gradients, training deep neural networks was challenging before ResNet. To add the outputs from an earlier layer to a later layer, ResNet employs skip connections. This aids it in overcoming the problem of vanishing gradients. Due to this, it could become possible to achieve several advancements in various computer vision tasks. Regardless of its success in generating gradient flow between building blocks, the additive function limits the opportunity to explore other potential complementary features because of the basic shortcut connecting technique. Ilija et al. in Radosavovic et al. (2020) introduced a regulator module as a memory mechanism to extract complementary features, which would then be sent to the ResNet. Convolutional recurrent neural networks are used in the regulator module because they have been demonstrated to be good at retrieving spatiotemporal information RegNetY320 is the name given to the new regulated networks. The regulator module is simple and may be added to any ResNet design. The non-bottleneck building block and the bottleneck building block are two types of ResNet building blocks proposed and show that RegNet outperformed on CIFAR10/100 and ImageNet datasets. Flowing are the equations (Eqs. (1)–(6)) that express the tth bottleneck RegNet module.

$$X_2^t = ReLU\left(BN\left(W_{12}^t * X_1^t + b_{12}^t\right)\right) \tag{1}$$

$$\left[H^t, C^t\right] = ReLU\left(BN\left(ConvLSTM\left(X_2^t, \left[H^{t-1}, C^{t-1}\right]\right)\right)\right) \tag{2}$$

$$X_3^t = ReLU\left(BN\left(W_{23}^t * X_2^t + b_{23}^t\right)\right) \tag{3}$$

$$X_4^t = ReLU\left(BN\left(W_{34}^t * Caoncat[X_3^t, H^t]\right)\right) \tag{4}$$

$$X_5^t = ReLU\left(BN\left(W_{45}^t * X_4^t + b_{45}^t\right)\right) \tag{5}$$

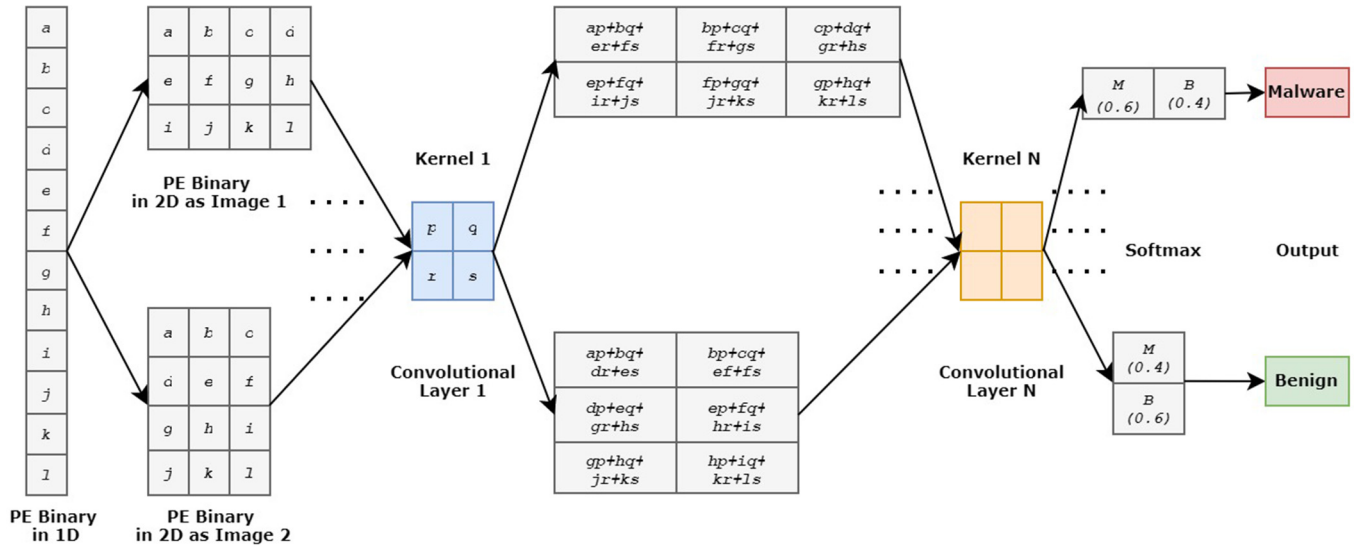


Fig. 3. A traditional malware detection task using a CNN-based model.

Table 5
Overview of pre-trained models.

Sr#	Model	Year	Depth	Layers	Size (MB)	Parameters (millions)	Non-trainable parameters	Trainable parameters
1	VGG16	2014	16	41	515	138	137,897,600	102,400
2	ResNet50	2015	50	177	96	25.6	25,548,800	51,200
3	InceptionV3	2015	48	316	89/ 87	23.9	23,848,800	51,200
4	VGG19	2014	19	47	535	20.1	20,024,384	25,089
5	MobileNet	2018	28	55	16	3.2	3,228,864	50,177
6	Xception	2013	71	171	88	22.9	20,861,480	100,353
7	DenseNet169	2016	169	338	57	12.7	12,642,880	81,537
8	DenseNet201	2017	201	709	80	18.4	18,321,984	94,081
9	InceptionResNetV2	2017	164	825	215	54.4	54,336,736	38,401
10	MobileNetV2	2018	53	155	13/ 14	3.5	3,468,000	32,000
11	ResNet152V2	2015	307	570	232	58.4	58,331,648	100,353
12	AlexNet	2012	8	25	227	61	60,897,600	102,400
13	SqueezeNet	2017	18	68	4.5	23.5	23,084,252	48,842
14	NasNetMobile	2018	389	914	23	4.4	4,269,716	51,745
15	RegNetY320	2020	320		553	145	144,754,400	103,840

$$X_1^{t+1} = ReLU(X_1^t + X_5^t) \tag{6}$$

W_{ij}^t is the convolutional kernel whose mapping feature maps X_i^t to X_j^t, and B_{ij}^t is the correlative bias. In which ReLU, Batch normalisation and concatenation layer functions are used. The Rectified linear activation function changes the node's summed weighted input into the node's activation or output for that input. Batch normalisation is a technique for training very deep neural networks that standardise each mini-input batch to a layer. This helps to stabilise the learning process and significantly reduces the number of training epochs needed to train deep networks.

3.2.2. Rationale behind the proposed framework

Fig. 3 depicts a conventional malware detection process using a learning-based model. Based on the training process, the final fully connected layer of the learning model returns a vector of logits. The vector then passes through the softmax activation function that assigns a certain probability to each classification label (malware and benign in our case). The softmax activation function is used on the last output layer of the pre-trained model for final malware detection. Fig. 1 illustrates the detailed conversion process of the PE file into an image. The width and length of a converted image vary and depend upon the content and size of the PE file. Fig. 3 illustrates that the binaries of a PE file can be mapped into the two 2D matrices differently.

The binaries of the same PE file are reshaped in two different images. Both images are passed through certain layers of the same learning model. The final logits vector of each image could be different. Hence, the softmax assigned different probabilities and eventually different outputs to the same PE file. The rationale behind this is that the last layer of the pre-trained model is being fine-tuned. The models do not properly learn the knowledge through transfer learning. Deep learning pre-trained models are not specifically designed for the malware detection task. Our proposed model handles this problem by extracting all the features from the last fully connected layer. Thus, the proposed framework does not depend on the probability assigned to each class by the learning model. Rather, we will use the machine learning model for the final malware detection. We have experimented with extracting features from the second fully connected and earlier layers. The results were not promising. The last fully connected results gave better results as it has more deep and rich features. The number of features returned by each model is different, depending on the density and architecture of each model. For example, the last fully connected layer of VGG16 models returned 4096 nodes. Hence, the vector size of 4096 is used to represent each image and later provided to the final detector for malware detection.

Secondly, a deep learning model can be used for classification and deep feature computation. In the first approach, the deep learning model is used end-to-end for feature computation and malware detection. In the second approach, the deep features are extracted from any layer and are given to a linear classifier for final malware detection. Our proposed model used the second approach. In the first approach, end-to-end, deep learning models require high computation resources and

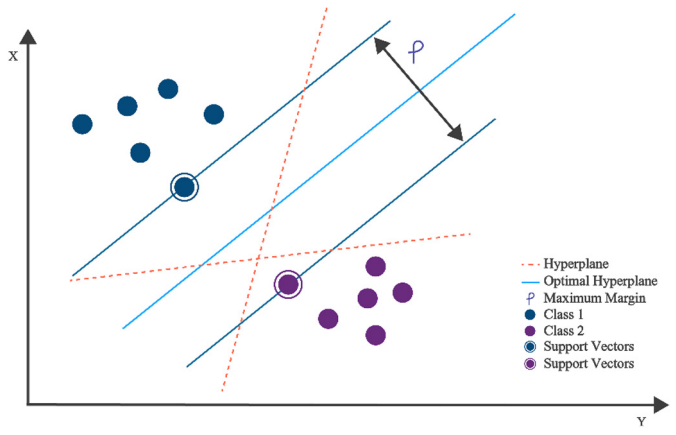


Fig. 4. Support vector machine.

parameter tuning using the new image dataset. Hence, the model may suffer from overfitting. Furthermore, it may require a lot of training data that may not be available in the case of malware detection. Lastly, we need a quick response to classify a PE file as malware or benign to avoid subsequent damage. We also need to reduce the reliance on knowledge from domain experts for reverse engineering tasks. Hence, our proposed malware detector is based on the second approach that addresses all these requirements. We use a linear classifier, i.e., Support Vector Machine (SVM), for final malware detection.

3.2.3. Final detector

The final detector is not involved in feature computation, thus reducing the possibility of overfitting the model. The SVM classifies and separates the two data classes based on the notation to the margin on either side of the hyperplane. Fig. 4 depicts the SVM.

The accuracy in classifying a data point can be maximised by increasing the margin and distances between hyperplanes. SVM can be classified into two major categories. It can be linear or non-linear based on the kernel function. It can also be one-class and multi-class based on detection type (Schölkopf et al., 2000). SVM needs training at different time intervals for better results to learn the dynamic user's behaviour. It can be used for classification and regression. It plots all the data points in space containing all the features and then segregates the classes (malware and benign in our case). The hyperplane, a.k.a. decision boundary, is oriented in such a way that the closest points of each class should have a maximum distance (φ) to it. Support vector points are those points that lie on the border of the hyperplane. The instances of class 1 and class 2 are separated on both sides of the hyperplane. The time complexity of SVM is $O(n^2)$, where n is the number of instances (Shaukat et al., 2020a). The performance of the proposed framework is validated with other machine learning classifiers. Table 6 presents the compact overview of various machine learning models used in this study to validate the performance of the proposed framework.

3.2.4. CNN-based SVM malware detector

Recently, researchers have been training CNN models for malware detection (Lad and Adamuthe, 2020). Most of these CNN-based models use softmax activation on the final layer. The softmax activation function is used for the final malware detection and to reduce the cross-entropy loss. In this sub-section, we have experimented by replacing the softmax layer with the linear support vector machine (SVM). SVM works to minimise the margin-based loss instead of cross-entropy. In our experiments, we have used L2-norm SVM as a final layer. CNN-SVM is extensively used in the literature for speech recognition and image classification tasks (Tang, 2013; Agarap, 2017). We have employed this approach to analyse the performance of CNN-based SVM for malware detection. In our experiments, we used L2-SVM instead of standard hinge loss. This approach is different to the proposed framework.

A support vector machine was initially proposed for binary classification (Cortes and Vapnik, 1995); however, multi-class SVM can be used for multi-class classification. Given a dataset $\mathcal{D}$ where $\mathcal{D} \in \{(x_1, y_1),(x_2,y_2),\ldots,(x_n, y_n)\}$, xi is the training sample and y_i is its corresponding label, target $\mathcal{T}_n \in \{1,-1\}$, ξ_n is the cost function that penalises the data points deviated from the marginal requirements of the hyperplane, c is the penalty parameter, $w^t w$ is the Manhattan norm (a.k.a. L1 norm), w is weight, b is the bias, the objective function $f(w,x) = (w.x + b)$ is used to find the optimal hyperplane that separate both classes (malware and benign). The model learns the optimal values of w and b using the constrained optimisation given below (Eqs. (7)–(9)):

$$min\ \frac{1}{N} w^T w + c \sum_{i=0}^{N} \xi_n \tag{7}$$

$$s.t.\ \mathcal{T}'_n(w.x + b) \geq 1 - \xi_n \tag{8}$$

$$\xi_n \geq \quad 0 \forall n \tag{9}$$

The unconstrained optimisation problem of Eq. (7) can be defined as Eq. (10):

$$min\ \frac{1}{N} w^T w + c \sum_{i=0}^{N} max(0, 1 - \mathcal{T}'_n(w^T\ x_i + b)) \tag{10}$$

where $(w^T\ x_i + b)$ is the predictor function and $\mathcal{T}'_n$ is the actual label.

The loss function for Eq. (10) is hinge loss. Eq. (11), also called the L2-SVM, provides a more stable result. It uses the $\|w\|_2^2$ Euclidean norm (a.k.a L2 norm) with the squared hinge loss.

$$min\ \frac{1}{N} \|w\|_2^2 + c \sum_{i=0}^{N} max(1 - \mathcal{T}'_n(w^T x_i + b), 0)^2 \tag{11}$$

The CNN model comprises four Convolutional, four MaxPooling, four Dropout, four Dense, and two Flatten layers. The model architecture is based on four iterations of layer sequence: Convolutional, MaxPooling, Dropout, and Dense. In the end, two flatten layers are used as fully connected layers before the final layer. Our approach involves preprocessing. To overcome the overhead of preprocessing, we used the dropout and DeCov regularisers. The dropout and DeCov regularisers improve the network convergence and significantly reduce the model overfitting (Cogswell et al., 2015). The last layer is modified with the L2 norm to make the final layer as SVM, as given in Eq. (11). The final prediction of a test sample x_t is made using Eq. (12).

$$arg_t max(w^T x_t)t \tag{12}$$

where $\arg_t max$ is a function that will return the indices of the highest scores for the test sample x_t among the t predicted classes. The proposed malware detector and CNN-SVM are different. CNN-SVM is a simple CNN model with few layers with L2-SVM loss. The proposed detector used transfer learning and pre-trained models as feature extractors and then a full SVM classifier as a final detector to classify benign and malware images. The performance of the proposed malware detector is also compared with CNN-SVM in the upcoming sections. The result section reports deep insights and detection effectiveness of the proposed framework.

4. Results and discussion

This section highlights some insights into the detection effectiveness of various malware detectors. This result section discusses malware detection from five perspectives. First, Section 4.1 considers the 15 deep learning models for end-to-end classification to detect malware using three different datasets. We also consider the imbalanced data problem in this sub-section. Various state-of-the-art augmentation techniques are used to handle the imbalanced data problem. The effect of these augmentation techniques on detection effectiveness is also

Table 6
A compact overview of machine learning models.

Model	Year	Ref. No	Description	Limitations
Decision Tree	1979	Ross (1993)	• Works on an if-then rule to find the best immediate node. • Continue the process until the predicted class is obtained.	• Difficult to change the data without affecting the overall structure. Complex, expensive and time-consuming.
Naïve Bayes	1960	Frank and Hall (2011)	• A probabilistic classifier that takes less computational time. • Assumes that a feature is entirely independent of all other present features.	• Assigns 0 probability if some category in the test data set is not present in the training data set. • Stores entire training examples • Needs massive data to obtain good results.
Random Forest	1995	Agrawal and Srikant (1995)	• Composed of many DTs. • Every DT yields a prediction. • The prediction that having a maximum number of votes will be the final prediction of the model.	• Computational cost is higher. • Slow prediction generator
Logistic Regression	1970	Akram et al. (2021)	• A.k.a., the logit model is a classification model. • Used a logistic function to model for modelling a binary classification. • Does not require a linear relationship between input and output variables.	• Suffer with restrictive expressiveness. • Struggle with the problem of complete separation. When a feature can separate each class perfectly, the LR model cannot be trained.
Linear Discriminator Analysis	1936	Maulana et al. (2022)	• Considers the linear combinations of variables that better explain the data. • It can work well with small sample sizes. • The decision boundary is linear. Provided the dimensionality reduction that later helped in feature engineering.	• Sensitive to outliers. • Too many assumptions and restrictions. • Linear decision boundary might not generalise well to all the classes and adequately separate both classes.
K-Nearest Neighbour	1951	Cunningham and Delany (2021)	• A non-parametric classifier that groups a data sample based on proximity. Similar points lie near each other in a sample cluster. • It is a lazy learner model that does not learn from the data immediately; rather first stores the dataset • It is adaptive and easy to implement.	• It is not scalable and does not perform well with high-dimension data. • It is computationally expensive. • A lower value of k tends to be overfitting, and a higher value tends to *smooth out*. It always required the value of k, which makes it complex sometimes.
Histogram-based Gradient Boosting	1990	Huo et al. (2022)	• Provides built-in support to handle missing values. • Bins the input samples into integer-based value bins.	• Needs a conversion of data values into integer-based to form bins.
Adaboost	1995	Galen and Steele (2021)	• Combines the inputs from a sequence of weak learners applied to repeatedly modify training dataset versions. • Combined output through voting. The weights are revised on each iteration. The learning model is reapplied to the revised weighted dataset. • It is Adaptive boosting.	• Computational expensive. • Cannot do parallelism for each learner. It needs to wait for the output from the previous learner. • Needs a balanced dataset for better performance. • Cannot perform well when the data is noisy or has outliers.
Gradient Boosting	1999	Gibert et al. (2022)	• Weak learners are ensembled to outperform in terms of efficiency and accuracy. • Typically, decision trees are the week learners. The outputs of decision trees are combined to achieve better results. • Uses a gradient to reduce the loss function.	• Does not provide parallelism. • Considers the loss from all possible tree splits to form a new branch that makes it inefficient. • Prone to overfitting. • Hard to interpret the final models.
XGBoost	2014	Al-Hashmi et al. (2022)	• Speedy and high-performance classifier for large datasets. • Performs parallelism with a single tree. • Uses advanced regularisation such as L1 and L2. • Also handles missing values in the dataset. • It is extreme Gradient Boosting.	• Does not perform well where the number of training samples is less than the number of features. • Requires a lot of resources to train larger datasets.
Bagging	1994	Kumar and Subbiah (2022)	• A.k.a. bootstrapping method. • The base models are executed on the bags of the whole dataset. A bag is a subset of the whole dataset with a replacement to balance the size of the whole dataset. • The same model is learned with multiple subsets of the dataset.	• Reduce the model interpretability. • Computational expensive. • The final model can experience the problem of high bias.

analysed. The models are fine-tuned on various hyperparameters for malware detection. Second, the performance of CNN-based SVM is analysed in Section 4.2. Third, the detection effectiveness of the proposed malware framework is presented in Section 4.3. The proposed

framework HD(LM)2D performance is validated with other machine learning models (when used as final detectors). In addition, this section also compares the performance of the proposed malware framework with other published work on the frequently used Malimg dataset. Due to privacy, the latest attack patterns are not available. Fourth, to consider the shortage of attack patterns problem, Section 4.4 validates the detection effectiveness of the proposed HD(LM)^{2}D on a smaller dataset. These results reveal the performance of the proposed malware detector when insufficient data samples are available for training. Fifth, the used image dataset (datasetA, datasetB, datasetC, and Malimg) are generated using all the features and binaries of good ware and malware PE files. The API call is one feature of the PE file. The API calls from all the PE files are extracted and converted into feature vectors. The API-based feature vectors are then transformed into an image-based API dataset. Here we compare the performance of detectors on an image-based dataset generated using to Section 4.5 compares the performance of detectors on an image-based dataset generated using one input feature (API call) or considering all features/binaries into images. We show that the proposed malware detector has outperformed state-of-the-art learning-based detectors on large and small datasets, including published static and dynamic techniques.

Different datasets are used in accordance to provide a true comparison of the state-of-the-art with the proposed framework. All the PEs are transformed into images using the aforementioned methodology. Various data pre-processing techniques are applied to scale and normalise the data as per the requirement of each deep learning model. The malware images are divided into training, validation, and testing sets with a ratio of 60:20:20, respectively. However, the data augmentation techniques are only applied to the training data to generalise the model in a better way. The 10-fold cross-validation is further used to validate the results. The statistical result comparison is made using the result of 10-fold cross-validation. For brevity of the comparison with the state-of-the-art, the same splitting mechanism is used. The experiments are performed with different values of epochs and batch sizes. The experiments are performed on the high-performance computing (HPC) system of The University of Newcastle (UON), Australia, having an Nvidia V100 GPU. Detection effectiveness includes accuracy, loss, true negative rate, true positive rate, false positive rate and false negative rate. Accuracy is the ratio of correctly classified malware and benign PEs to all the malware and benign PEs in the dataset. The accuracy values are given in percentage. The loss, on the other hand, is not a percentage. It is a summation of the differences/errors made by the model against each PE file in the validation or test set. Specificity, a.k.a. true negative rate (TNR), is the ratio between correctly classified negative samples to the total negative samples. Sensitivity, a.k.a. true positive rate (TPR), is the ratio between correctly classified positive samples to the total positive samples. False positive rate (FPR) is the ratio of the incorrectly classified negative samples as positive to the total number of negative samples. False negative rate (FNR) is the ratio of incorrectly classified positive samples as negative to the total number of positive samples (Shaukat et al., 2020c).

4.1. Detection effectiveness on imbalanced datasets

This section provides the detection effectiveness of 15 fine-tuned deep learning-based malware detectors. Deep learning models are not specifically designed for malware detection. Using transfer learning, we have fine-tuned the 15 deep learning-based models for the malware detection task using end-to-end classification. Section 4.1 highlighted the problem of an imbalanced dataset, where the count of instances of each classification class is not balanced. As a result, the detector cannot generalise each class instance. Various state-of-the-art augmentation techniques are used to balance the data. Tables 7, 8 and 9 provide the detection effectiveness of 15 fine-tuned deep learning-based malware detectors on three different datasets. The analysis is based on loss and accuracy. The analysis is performed with batch sizes 8, 16, and 32. The effect of various epoch values is also analysed. These deep learning models are selected based on their best performance in the literature on image and malware detection tasks. A total of 20,000 malicious PEs and 10,000 benign PEs corpus have been collected.[1] The malware corpus is the collection of malicious PEs from the Microsoft malware dataset (Microsoft, 2015), Malimg (Nataraj et al., 2011), and VirusShare (VirusShare, 2022). The benign PEs are collected from multiple online resources. The collected corpus is further divided into three datasets: datasetA, datasetB, and datasetC. The datasets are generated to analyse the performance of learning-based detectors on different dataset sizes.

Table 7 presents the results of 15 fine-tuned deep learning-based malware detectors on datasetA. DatasetA consists of 2000 malware and 10,000 benign PEs. The samples are divided with a ratio of 60:20:20 for training, validation, and testing, respectively. That is, 1200 malware and 6000 benign PEs are used for training, 400 malicious and 2000 benign PEs for validation, and 400 malicious and 2000 benign PEs for testing. All the PEs are transformed into images. We see that the VGG16 model has the highest accuracy of 92.24% for the batch size of 64 and 350 epochs. The test loss value is <1 for all settings of hyperparameters for both validation and test data. The ResNet50 has given the highest loss value of 46.7. The model did not generalise all the malicious PEs images. The model has overfitted the benign PEs. This might happen due to the low instances of the malicious class. Additionally, it is worth noting that the given test and validation accuracy values in Tables 7, 8, and 9 are based on the ratio of correctly classified malicious and benign PEs.

However, the balanced accuracy value for ResNet50 is important to discuss. Balanced accuracy (BA) is an important metric in the case of an imbalanced dataset. It is the average value of recall and specificity for each class (malware and benign in our case). Hence, $BA = \frac{\text{True Positive Rate}+\text{True Negative Rate}}{2}$ is the average of correctly classified benign and correctly classified malware samples. For example, the test accuracy value is 68.32% for 150 epochs and 8 batch sizes. The ResNet50 did not generalise the malicious samples perfectly. The recall (a.k.a. true positive rate) for the benign class is 79.8% (1596 samples are correctly classified out of 2000) and 11% (only 44 samples are classified correctly out of 400). The balanced accuracy of ResNet50 is 45.4% ((79.8 + 11)/2). Overall, ResNet50 gives the worst performance. The VGG19 has a comparative accuracy of 92.19%, with a lower loss value for both validation and test data. The bold value is the best performance of each model. The bold value with a yellow background is the best among all models.

Table 8 presents the results of 15 fine-tuned deep learning-based malware detectors on datasetB. DatasetB consists of 20,000 malware and 10,000 benign PEs. The samples are divided with a ratio of 60:20:20 for training, validation, and testing, respectively. 12,000 malware and 6000 benign PEs are used for training, 4000 malicious and 2000 benign PEs for validation, and 4000 malicious and 2000 benign PEs for testing. The VGG16 model has again outperformed with an accuracy of 93.58% and a lower loss rate. ResNet50 model gave a higher loss value of 54.62 for test data. VGG19 has an accuracy value of 93.04%. ResNet50 again did not generalise the samples of minority classes and skewed towards the majority class. The overall accuracy of ResNet50 is 44.58%, and the balanced accuracy is 38.33%, with 150 epochs and 32 batch size. The true positive rate is 57.02%. Out of 4000 malicious PEs, only 2281 are correctly classified. On the other hand, the true negative rate is 19.65% for benign samples. 393 benign samples are correctly classified out of 2000. The bold value is the best performance of each model. The bold value with a yellow background is the best among all the models.

Table 9 presents the results of 15 fine-tuned deep learning-based malware detectors on datasetC. DatasetC consists of 20,000 malware and 10,000 benign PEs. All the samples are the same as in datasetB.

[1] The code and datasets are available by request.

Table 7
Detection effectiveness of malware detectors on datasetA.

Sr. No.	Model	Epochs	Batch Size	Valid Loss	Valid Acc.	Test Loss	Test Acc.
1	VGG16	150	8	0.9573	78.19%	0.6878	89.08%
		150	16	0.6512	85.78%	0.4279	90.86%
		150	32	0.578	86.99%	0.4075	90.71%
		250	32	0.4898	89.16%	0.4279	91.93%
		150	64	0.621	82.78%	0.3445	91.17%
		250	64	0.4914	88.14%	0.338	**92.24%**
2	ResNet50	150	8	46.63	69.39%	46.7	68.32%
		70	8	21.5819	69.39%	21.57	69.33%
		150	16	21.85	69.39%	18.43	69.18%
		150	32	30.88	69.39%	31.42	**69.38%**
3	InceptionV3	150	8	6.86	67.22%	4.68	67.14%
		150	16	4.66	61.67%	4.52	62.04%
		250	16	3.29	74.11%	3.48	72.19%
		150	32	5.55	71.30%	3.599	**72.55%**
4	VGG19	150	8	0.8179	86.86%	0.3869	90.91%
		150	16	0.666	86.80%	0.424	**92.19%**
		150	32	0.5474	88.46%	0.3023	89.84%
5	MobileNet	150	8	16.5933	59.82%	9.54	**74.08%**
		150	16	14.74	57.21%	8.89	66.02%
		250	16	11.15	66.65%	8.348	73.77%
		150	32	10.6329	55.99%	5.44	70.25%
6	DenseNet169	150	8	7.11	77.49%	6.56	81.07%
		150	16	5.72	75.26%	5.81	80.71%
		150	32	4.53	77.10%	4.702	77.95%
		250	32	4.65	77.55%	4.08	**81.78%**
7	Xception	150	8	4.94	65.43%	5.84	57.44%
		150	16	6.402	50.64%	7.04	46.17%
		150	32	3.24	65.43%	3.88	**62.04%**
8	DenseNet201	150	8	14.19	58.61%	20.38	52.80%
		150	16	11.86	59.38%	11.83	66.22%
		150	32	9.74	73.02%	10.99	**75.05%**
		250	32	7.7311	73.53%	7.07	67.75%
		150	64	9.5959	46.36%	7.156	55.05%
		250	64	6.1271	60.97%	9.188	62.34%
9	Inception ResNet V2	150	8	5.035	65.37%	6.066	**65.91%**
		150	16	8.3494	47.39%	8.959	46.17%
		150	32	3.389	61.54%	4.63	52.80%
10	MobileNetV2	150	8	12.5227	67.22%	13.35	69.81%
		150	16	10.577	63.39%	10.911	66.53%
		150	32	12.3141	70.34%	10.84	**70.81%**
11	ResNet152V2	150	8	13.9263	73.34%	12.98	74.64%
		150	16	12.24	75.06%	9.25	**77.91%**
		150	32	11.49	74.87%	9.099	77.44%
12	NasNetMobile	150	8	6.04	70.60%	4.226	76.29%
		150	16	6.94	52.04%	6.202	54.03%
		150	32	4.307	56.17%	5.53	50.70%
		250	32	2.8907	70.41%	3.01	70.96%
		150	64	2.3857	75.32%	2.3213	**77.29%**
		250	64	3.666	64.16%	2.7905	65.00%
13	AlexNet	150	8	10.23	70.13%	8.533	69.52%
		150	16	10.48	68.48%	9.12	**70.32%**
		150	32	9.03	66.72%	8.49	68.54%
14	SqueezeNet	150	8	10.89	63.46%	10.29	64.28%
		150	16	10.39	66.41%	8.18	67.21%
		150	32	10.88	71.23%	8.38	**70.27%**
15	RegNetY320	150	8	4.12	76.34%	3.19	78.59%
		150	16	3.92	79.17%	3.57	**79.24%**
		150	32	2.98	78.18%	2.41	78.92%

The 12,000 malicious PEs are used for training, 4000 for validation and 4000 for testing. The benign PEs are divided with a ratio of 60:20:20. Hence, 6000 for training, 2000 for validation and 2000 for testing. Conspicuously, the count of benign and malicious PEs for training

Table 8
Detection effectiveness of malware detectors on datasetB.

Sr. No.	Model	Epochs	Batch Size	Valid Loss	Valid Acc.	Test Loss	Test Acc.
1	VGG16	150	8	0.9236	89.45%	0.564	92.54%
		150	16	0.7045	89.39%	0.4013	93.56%
		150	32	0.6954	88.09%	0.487	90.75%
		250	32	0.5649	91.09%	0.3765	92.22%
		150	64	0.36	91.69%	0.2609	**93.58%**
		250	64	0.5372	90.09%	0.3284	92.69%
2	ResNet50	150	8	28.91	46.80%	19.89	54.38%
		70	8	64.94	38.07%	31.41	39.53%
		150	16	33.37	41.98%	16.07	**64.51%**
		150	32	67.29	35.00%	54.62	44.58%
3	InceptionV3	150	8	8.01	74.09%	5.98	78.97%
		150	16	4.21	75.16%	2.57	79.12%
		250	16	3.89	76.94%	3.1	78.20%
		150	32	4.01	75.38%	2.72	**80.23%**
4	VGG19	150	8	0.791	91.16%	0.5591	91.34%
		150	16	0.669	90.56%	0.5262	91.98%
		150	32	0.5344	90.34%	0.3785	**93.04%**
5	MobileNet	150	8	11.74	76.25%	10.26	74.31%
		150	16	15.04	74.67%	8.23	73.79%
		250	16	14.64	75.28%	8.3	74.41%
		150	32	8.503	77.42%	5.15	**76.04%**
6	DenseNet169	150	8	18.53	72.70%	10.53	83.57%
		150	16	11.59	76.03%	7.823	82.34%
		150	32	11.43	73.42%	5.94	**83.59%**
		250	32	14.31	72.33%	8.71	80.87%
7	Xception	150	8	6.68	73.36%	5.18	77.96%
		150	16	4.52	76.95%	3.56	**80.29%**
		150	32	3.53	76.11%	4.29	73.93%
8	DenseNet201	150	8	24.64	70.58%	25.67	75.98%
		150	16	17.97	70.98%	16.3	77.62%
		150	32	15.84	68.50%	14.24	76.23%
		250	32	12.73	73.23%	12.18	**78.31%**
		150	64	11.03	68.82%	10.38	76.54%
		250	64	11.12	71.52%	12.39	76.28%
9	Inception ResNet V2	150	8	8.71	72.11%	10.86	73.62%
		150	16	4.99	73.33%	5.77	73.28%
		150	32	4.27	74.41%	5.29	**75.35%**
10	MobileNetV2	150	8	10.73	73.36%	10.42	73.35%
		150	16	9.23	73.42%	8.73	**75.18%**
		150	32	8.61	72.05%	9.12	73.97%
11	ResNet152V2	150	8	22.52	79.33%	14.63	81.04%
		150	16	17.7	77.31%	12.19	**81.09%**
		150	32	16.02	75.86%	11.71	80.21%
12	NasNetMobile	150	8	6.2	76.55%	6.34	79.45%
		150	16	4.11	76.81%	4.15	**78.46%**
		150	32	3.77	76.17%	3.52	78.43%
		250	32	3.33	75.61%	4.43	76.60%
		150	64	2.9	74.45%	3.09	76.54%
		250	64	3.66	73.53%	4.03	75.91%
13	AlexNet	150	8	9.91	72.87%	8.34	72.52%
		150	16	10.02	70.76%	8.59	71.35%
		150	32	10.64	72.65%	8.02	**73.11%**
14	SqueezeNet	150	8	9.68	72.76%	9.47	74.51%
		150	16	9.12	73.71%	7.85	**75.24%**
		150	32	9.63	71.87%	8.26	74.95%
15	RegNetY320	150	8	2.34	80.18%	2.21	81.07%
		150	16	3.14	82.65%	2.76	**82.71%**
		150	32	2.83	83.12%	1.93	82.54%

Table 9
Detection effectiveness of malware detectors on datasetC.

Sr. No.	Model	Epochs	Batch Size	Valid Loss	Valid Acc.	Test Loss	Test Acc.
1	VGG16	150	8	0.7649	90.41%	0.427	93.12%
		150	16	0.4896	91.72%	0.3067	93.84%
		150	32	0.4312	91.72%	0.2509	93.68%
		250	32	0.4069	91.76%	0.31	92.44%
		150	64	0.3291	92.23%	0.2173	**94.44%**
		250	64	0.459	91.11%	0.26	93.95%
2	ResNet50	150	8	14.24	74.71%	15.97	**74.69%**
		70	8	18.05	74.69%	17.56	74.61%
		150	16	19.76	72.81%	16.01	74.56%
		150	32	28.88	70.97%	30.54	74.84%
3	InceptionV3	150	8	5.07	78.39%	4.51	78.99%
		150	16	4.08	76.13%	2.36	79.26%
		250	16	3.22	78.72%	2.63	**80.76%**
		150	32	2.87	77.33%	1.9	80.46%
4	VGG19	150	8	0.6398	91.39%	0.3659	92.95%
		150	16	0.4814	91.11%	0.3327	93.53%
		150	32	0.4998	91.39%	0.2789	**93.64%**
5	MobileNet	150	8	10.59	76.69%	9.43	**78.22%**
		150	16	10.97	74.80%	7.43	74.49%
		250	16	10.27	75.99%	8.28	76.58%
		150	32	7.58	78.24%	4.39	76.13%
6	DenseNet169	150	8	6.71	81.08%	4.86	83.85%
		150	16	4.64	82.45%	4.94	84.19%
		150	32	3.71	91.12%	2.66	84.15%
		250	32	4.48	81.75%	3.23	**84.89%**
7	Xception	150	8	4.72	74.64%	3.61	78.15%
		150	16	3.7	77.43%	3.17	**81.45%**
		150	32	2.8	76.57%	3.41	74.21%
8	DenseNet201	150	8	11.84	79.82%	11.6	79.80%
		150	16	11.76	77.16%	11.7	77.83%
		150	32	9.09	77.13%	8.84	77.44%
		250	32	6.17	81.54%	5.57	**82.38%**
		150	64	5.36	78.21%	4.47	79.84%
		250	64	5.59	75.38%	8.95	76.57%
9	Inception ResNet V2	150	8	4.01	75.78%	5.95	**76.77%**
		150	16	3.34	75.38%	4.24	74.36%
		150	32	3.38	76.69%	4.21	75.55%
10	MobileNetV2	150	8	8.82	73.80%	9.7	74.42%
		150	16	8.65	74.02%	7.67	**75.28%**
		150	32	8.51	73.04%	8.8	74.01%
11	ResNet152V2	150	8	11.35	80.31%	11.19	81.99%
		150	16	11.05	77.71%	8.59	**82.41%**
		150	32	9.5	76.62%	8.95	80.31%
12	NasNetMobile	150	8	4.54	78.46%	3.43	**81.99%**
		150	16	3.02	79.00%	2.66	79.41%
		150	32	2.33	77.86%	1.93	79.62%
		250	32	2.14	75.90%	2.25	76.65%
		150	64	1.84	79.40%	1.54	80.20%
		250	64	2.72	77.72%	2.47	78.33%
13	AlexNet	150	8	9.25	75.95%	7.52	77.18%
		150	16	9.85	74.17%	8.36	76.41%
		150	32	8.81	76.01%	7.14	**77.48%**
14	SqueezeNet	150	8	9.56	72.95%	8.52	78.32%
		150	16	8.92	73.84%	7.51	79.87%
		150	32	9.51	73.51%	7.94	**81.28%**
15	RegNetY320	150	8	2.21	84.52%	1.75	87.95%
		150	16	2.87	84.96%	1.24	88.01%
		150	32	1.96	85.41%	1.19	**88.52%**

Table 10
A comparison of detection effectiveness of fine-tuned deep learning-based malware detectors with CNN-based SVM malware detector on datasetA.

Sr. No.	Model	Epochs	Batch size	Valid loss	Valid Acc.	Test loss	Test Acc.
1	VGG16	150	32	0.578	86.99%	0.4575	90.71%
2	ResNet50	150	32	30.88	69.39%	31.42	69.38%
3	InceptionV3	150	32	5.55	71.30%	3.599	72.55%
4	VGG19	150	32	0.5874	85.46%	0.4613	89.84%
5	MobileNet	150	32	10.6329	55.99%	5.44	70.25%
6	DenseNet169	150	32	4.53	77.10%	4.702	77.95%
7	Xception	150	32	3.24	65.43%	3.88	62.04%
8	DenseNet201	150	32	9.74	73.02%	10.99	75.05%
9	Inception ResNet V2	150	32	3.389	61.54%	4.63	52.80%
10	MobileNetV2	150	32	12.3141	70.34%	10.84	70.81%
11	ResNet152V2	150	32	11.49	74.87%	9.099	77.44%
12	NasNetMobile	150	32	4.307	56.17%	5.53	50.70%
13	AlexNet	150	32	9.03	66.72%	8.49	68.54%
14	SqueezeNet	150	32	10.88	71.23%	8.38	70.27%
15	RegNetY320	150	32	2.98	78.18%	2.41	78.92%
16	CNN-SVM	150	32	0.397	88.65%	0.4366	**91.17%**

is highly imbalanced. We have used the augmentation techniques mentioned in Section 4.1 to augment the benign training data to Section 3.1.3. The total number of benign PEs for training is 12,000. The benign training set includes 6000 real benign samples and 6000 augmented benign images.

Whereas the count of validation and testing benign PEs is the same. The augmentation is only performed on the training data. VGG16 has outperformed with an accuracy of 94.44%. VGG19 takes second place with an accuracy of 93.64%. The ResNet50 loss has also been decreased. After the augmentation, the results are improved. Each model has generalised the samples of both classes correctly. The overall accuracy of ResNet50 is 74.84%, and the balanced accuracy is 71.73%. The augmentation has significantly increased the accuracies for both classification classes. The true positive rate is 81.02% for the malware class, which has improved to 24% than without augmentation. Out of 4000 malicious samples, 3241 samples are correctly classified.

On the other hand, 1249 benign samples are correctly classified out of 2000. The improvement is 42.80% more than without augmentation. The true negative rate was 19.65% without augmentation for the same settings of ResNet50 in Table 8. The bold value is the best performance of each model. The bold value with a yellow background is the best among all models.

Overall, the accuracy of VGG16 is improved by roughly 1% for all hyperparameter settings. Surprisingly, ResNet50's accuracy values are enhanced by more than 10% for each setting of hyperparameters. With balanced data, the densest architecture RegNetY320 accuracy is improved by more than 5%. The VGG19 model's accuracy is significantly improved by >1% for batch sizes 8 and 16. The augmentation has increased the 3.49% accuracy on average for all the models. It is observed that the VGG16 model's accuracy decreases by increasing the number of epochs with a batch size of 32. However, the InceptionV3 accuracy is increased to 80.76% from 79.26% by increasing the number of epochs with 16 batch size. The accuracy of DenseNet201 is also increased from 77.44% to 82.38% by increasing the number of epochs from 150 to 250 for 32 batch size. On the other hand, the accuracy is decreased by 3% when the number of epochs is increased from 150 to 250 for batch size 64. A similar case happened with VGG16 with 64 batch size. The augmentation techniques proved efficient in increasing the accuracy of fine-tuned models and decreasing the loss values. During the analysis, we cannot exactly comment on the effect of hyperparameters. Sometimes, the increases in the batch size improve the accuracy and sometimes not. The same is valid for the number of epochs. We will provide the data and code on a reasonable request. We plan to analyse the effects of other hyperparameters and models in future.

4.2. Analysis of CNN-based SVM malware detector

This section provides a comparative analysis of CNN-SVM with deep learning-based models for malware detection. A detailed discussion of CNN-SVM is provided in Section 3.2.4. Three sets of experiments were performed on three datasets: datasetA, datasetB, and datasetC.

The same datasets are used to validate the performance of this approach. We have observed that better results are achieved with 150 epochs and 32 batch size. For brevity, we have used only this hyperparameter setting for the analysis in this section. Tables 10, 11, and 12 present the results of these datasets. It is seen that CNN-SVM has outperformed the existing approach on all three datasets. The experiments were performed using TensorFlow with 150 epochs, 32 batch size, 1e−3 learning rate, and 0.85 dropout rate.

The CNN-SVM has outperformed with an accuracy of 91.17% on datasetA, 93.39% on datasetB, and 95.15% on datasetC. The CNN-SVM has outperformed VGG16 with 0.46%, as demonstrated in Table 8. However, the accuracy of VGG19 is highest on datasetB with these hyperparameters settings as 93.04% in Table 11. CNN-SVM has increased the accuracy to 0.35%. Table 12 presents the results of datasetC, which was augmented to balance the data. The VGG16 gave 93.68% accuracy. CNN-SVM outperformed with 95.15%, which is 1.47% higher. The CNN-SVM outperformed other models with slightly higher accuracy. However, the increase is not significant though the training process took a shorter time than the training of fine-tuned deep learning models using transfer learning.

4.3. Detection effectiveness of the proposed malware detection

This section highlights the detection effectiveness of our proposed hybrid framework. The proposed framework extracts the deep features using a fine-tuned deep learning model. The extracted features are fed into a machine learning model to fully utilise the inherent connections between input and output. The proposed framework uses the SVM as a final detector. The densest deep learning model RegNetY320 outperformed with a combination of SVM for malware detection. We are unaware of any work using a RegNetY320 as a feature extractor or for end-to-end malware detection. We are also unaware of any work using learning-based models as feature extractors and SVM as a final detector. Hence, the proposed framework is novel and scalable. In principle, any deep learning and machine learning models can be used as feature extractors and final detectors, respectively.

It seems that augmentation has increased the malware detection effectiveness and reduced the training and testing loss in our previous experiments. For brevity, we have used datasetC to analyse the detection effectiveness of our proposed framework. All the experiments are conducted with 32 batch size and 150 epochs. Table 13 not only demonstrates the detection effectiveness of our proposed framework but also validates the results with state-of-the-art machine learning models. The model column lists all the models. The accuracy column

Table 11
A comparison of detection effectiveness of fine-tuned deep learning-based malware detectors with CNN-based SVM malware detector on datasetB.

Sr. No.	Model	Epochs	Batch size	Valid loss	Valid Acc.	Test loss	Test Acc.
1	VGG16	150	32	0.6954	88.09%	0.487	90.75%
2	ResNet50	150	32	67.29	50.00%	54.62	74.58%
3	InceptionV3	150	32	4.01	75.38%	2.72	80.23%
4	VGG19	150	32	0.5344	90.34%	0.3785	93.04%
5	MobileNet	150	32	8.503	77.42%	5.15	76.04%
6	DenseNet169	150	32	11.43	73.42%	5.94	83.59%
7	Xception	150	32	3.53	76.11%	4.29	73.93%
8	DenseNet201	150	32	15.84	68.50%	14.24	76.23%
9	Inception ResNet V2	150	32	4.27	74.41%	5.29	75.35%
10	MobileNetV2	150	32	8.61	72.05%	9.12	73.97%
11	ResNet152V2	150	32	16.02	75.86%	11.71	80.21%
12	NasNetMobile	150	32	3.77	76.17%	3.52	78.43%
13	AlexNet	150	32	10.64	72.65%	8.02	73.11%
14	SqueezeNet	150	32	9.63	71.87%	8.26	74.95%
15	RegNetY320	150	32	2.83	83.12%	1.93	82.54%
16	CNN-SVM	150	32	0.4544	92.00%	0.3373	**93.39%**

Table 12
A comparison of detection effectiveness of fine-tuned deep learning-based malware detectors with CNN-based SVM malware detector on datasetC.

Sr. No.	Model	Epochs	Batch size	Valid loss	Valid Acc.	Test loss	Test Acc.
1	VGG16	150	32	0.4312	91.72%	0.2509	93.68%
2	ResNet50	150	32	28.88	70.97%	30.54	74.84%
3	InceptionV3	150	32	2.87	77.33%	1.9	80.46%
4	VGG19	150	32	0.4998	91.39%	0.2789	93.64%
5	MobileNet	150	32	7.58	78.24%	4.39	76.13%
6	DenseNet169	150	32	3.71	91.12%	2.66	84.15%
7	Xception	150	32	2.8	76.57%	3.41	74.21%
8	DenseNet201	150	32	9.09	77.13%	8.84	77.44%
9	Inception ResNet V2	150	32	3.38	76.69%	4.21	75.55%
10	MobileNetV2	150	32	8.51	73.04%	8.8	74.01%
11	ResNet152V2	150	32	9.5	76.62%	8.95	80.31%
12	NasNetMobile	150	32	2.33	77.86%	1.93	79.62%
13	AlexNet	150	32	8.81	76.01%	7.14	77.48%
14	SqueezeNet	150	32	9.51	73.51%	7.94	81.28%
15	RegNetY320	150	32	1.96	85.41%	1.19	88.52%
16	CNN-SVM	150	32	0.2783	93.70%	0.2303	**95.15%**

presents the detection effectiveness of each fine-tuned deep learning model in this hyperparameter setting. The follow-up columns mention the results of each machine learning-based model when used as the final detector. Finally, the last column shows the performance of our proposed framework with bold green text.

Table 13 demonstrates the in-depth analysis of 15 fine-tuned deep learning-based models for malware detection. Each Cartesian value presents the framework's accuracy if the features are extracted using a fine-tuned deep learning model (listed in the model column) and fed to any machine learning model (listed in all the columns) as the final detector. For example, the accuracy column shows that if we apply a fine-tuned deep learning model for end-to-end malware detection, the obtained accuracy is 93.68%. However, if we extract the features using VGG16 and apply bagging as a detector, the obtained accuracy is 93.41%. The same is the case with the rest of the columns as a final detector.

The proposed framework improves the detection effectiveness of all the fine-tuned models. Though the accuracy of some fine-tuned models (when applied end-to-end) was even lower than 70%, the framework significantly increased the detection effectiveness of all fine-tuned models. We observed that the framework performed better with the densest architecture, i.e., RegNetY320. Table 14 provides a compact overview of the performance improvement compared to the fine-tuned deep learning-based malware detector. Bagging, Adaboost, Gradient Boosting, and XGBoosting have increased the accuracy for most models but not all. The HistGradient Boosting (HGB) classifier has outperformed all the fine-tuned models and given a 15.07% average increase to all the fine-tuned models. The proposed framework has given 16.55% average increase for to all of the fine-tuned models. The proposed framework significantly gave a 1.48% better performance than the HistGradient Boosting classifier. Logistic regression (LR) outperformed all fine-tuned models, though the average increase with LR is 16.05% which is 1% better than HGB. Random Forest (RF) has also performed better for all the deep learning models.

The average increase in accuracy is 14.11% which is <2.44%, than our proposed framework. Naïve Bayesian (NB), Decision Tree (DT), and Linear Discriminator Analysis (LDA) could not outperform all the models, though the average increase was 4.27%, 8.27%, and 12.4%, respectively. K-Nearest Neighbour (K-NN) has outperformed all fine-tuned models with an average increase of 14.4%, which is <2.15% than the proposed framework. Resultantly, Bagging, Adaboost, Gradient Boosting, XGBoosting, Naïve Bayesian, Decision Tree, and Linear Discriminator Analysis could not increase accuracy for all the models. Conversely, KNN, RF, LR and HGB have outperformed the fine-tuned models. Our proposed framework has outperformed not only the existing fine-tuned models but also the state-of-the-art machine learning classifiers.

Regarding models, VGG16, VGG19, and RegNetY320 are the models that could not outperform all the combinations of machine learning models (as final detectors) than fine-tuned model's performance individually (as end-to-end malware detector). For example, the end-to-end malware detection effectiveness of VGG19 is better than when the framework used Adaboost, gradient boosting, naïve bays, decision tree, and linear discriminator analysis classifiers as the final detectors. The generalisation capability of the proposed framework is accessed in Table 15. Table 15 demonstrates the comparison of our framework with

Table 13
A deep analysis of proposed framework (all values are in percentage %).

Sr. No	Model	Accuracy	Bagging	Adaboost	Gradient Boosting	XGBoost	HistGradient Boosting	Logistic Regression	Random Forest	Naïve Bays	Decision Tree	K-Nearest Neighbour	Linear Discriminator Analysis	Proposed framework
1	VGG16	93.68	93.41	92.11	91.81	93.59	96.12	97.23	95.31	84.87	90.31	96.72	89.21	**97.91**
2	ResNet50	74.84	93.2	92.29	90.23	93.18	96.04	96.53	94.39	81.51	89.37	94.78	84.43	**96.87**
3	InceptionV3	80.46	91.61	90.33	89.49	92.34	94.31	95.21	92.37	83.82	84.41	92.12	93.81	**96.02**
4	VGG19	93.64	95.23	93.48	93.58	94.28	96.52	97.36	95.85	86.43	91.57	96.53	90.53	**97.82**
5	MobileNet	76.13	93.1	92.14	91.33	92.57	96.33	97.24	95.89	86.57	88.43	95.47	95.81	**97.34**
6	Xception	74.21	91.48	90.23	91.13	93.31	95.11	96.31	94.71	82.89	88.81	94.58	95.77	**96.74**
7	DenseNet169	84.15	92.93	89.99	92.44	94.73	96.27	97.43	95.77	85.23	90.21	96.81	96.79	**97.55**
8	DenseNet201	77.44	90.23	87.81	89.23	94.18	96.51	97.78	95.56	86.57	89.9	96.51	97.23	**97.91**
9	InceptionResNetV2	75.55	91.44	90.25	90.93	94.03	96.48	97.41	94.97	87.1	88.53	95.91	96.14	**97.9**
10	MobileNetV2	74.01	92.51	91.14	91.82	93.81	96.43	97.35	95.35	87.41	89.43	95.58	95.44	**97.5**
11	NasNetMobile	79.62	93.14	92.1	91.22	93.48	95.93	96.33	94.56	84.86	88.33	95.21	95.23	**96.84**
12	ResNet152V2	80.31	93.22	90.98	91.58	92.13	95.22	96.24	94.23	85.75	88.58	92.47	95.35	**96.73**
13	AlexNet	77.48	89.97	88.54	90.27	90.85	94.28	95.28	93.43	83.29	88.56	93.86	86.59	**96.25**
14	SqueezeNet	81.28	90.72	88.96	91.55	91.67	95.08	95.94	94.27	83.84	89.18	94.28	88.51	**97.4**
15	RegNetY320	88.52	94.6	92.23	91.58	93.98	96.87	98.82	96.41	85.33	88.81	96.61	96.61	**98.93**

Table 14
A compact comparison of the detection effectiveness of proposed framework with state-of-the-art machine learning models (all values are in percentage % - green colour shows the % increment to the benchmark, i.e., accuracy column values).

Sr. No	Model	Accuracy	Bagging	Adaboost	Gradient Boosting	XGBoost	HistGradiant Boosting	Logistic Regression	Random Forest	Naïve Bays	Decision Tree	K-Nearest Neighbour	Linear Discriminator Analysis	Proposed Framework
1	VGG16	93.68	-0.27	-1.57	-1.87	-0.09	2.44	3.55	1.63	-8.81	-3.37	3.04	-4.47	**4.23**
2	ResNet50	74.84	18.36	17.45	15.39	18.34	21.2	21.69	19.55	6.67	14.53	19.94	9.59	**22.03**
3	InceptionV3	80.46	11.15	9.87	9.03	11.88	13.85	14.75	11.91	3.36	3.95	11.66	13.35	**15.56**
4	VGG19	93.64	1.59	-0.16	-0.06	0.64	2.88	3.72	2.21	-7.21	-2.07	2.89	-3.11	**4.18**
5	MobileNet	76.13	16.97	16.01	15.2	16.44	20.2	21.11	19.76	10.44	12.3	19.34	19.68	**21.21**
6	Xception	74.21	17.27	16.02	16.92	19.1	20.9	22.1	20.5	8.68	14.6	20.37	21.56	**22.53**
7	DenseNet169	84.15	8.78	5.84	8.29	10.58	12.12	13.28	11.62	1.08	6.06	12.66	12.64	**13.4**
8	DenseNet201	77.44	12.79	10.37	11.79	16.74	19.07	20.34	18.12	9.13	12.46	19.07	19.79	**20.47**
9	InceptionResNetV2	75.55	15.89	14.7	15.38	18.48	20.93	21.86	19.42	11.55	12.98	20.36	20.59	**22.35**
10	MobileNetV2	74.01	18.5	17.13	17.81	19.8	22.42	23.34	21.34	13.4	15.42	21.57	21.43	**23.49**
11	NasNetMobile	79.62	13.52	12.48	11.6	13.86	16.31	16.71	14.94	5.24	8.71	15.59	15.61	**17.22**
12	ResNet152V2	80.31	12.91	10.67	11.27	11.82	14.91	15.93	13.92	5.44	8.27	12.16	15.04	**16.42**
13	AlexNet	77.48	12.49	11.06	12.79	13.37	16.8	17.8	15.95	5.81	11.08	16.38	9.11	**18.77**
14	SqueezeNet	81.28	9.44	7.68	10.27	10.39	13.8	14.66	12.99	2.56	7.9	13	7.23	**16.12**
15	RegNetY320	88.52	6.08	3.71	3.06	5.46	8.35	10.3	7.89	-3.19	0.29	8.09	8.09	**10.41**
Average Increase			**11.70**	**10.08**	**10.46**	**12.45**	**15.08**	**16.08**	**14.12**	**4.28**	**8.21**	**14.41**	**12.41**	**16.56**

Table 15
A comparison of the proposed framework with the state-of-the-art.

Sr. No.	year	Dataset	Reference#	Models	Accuracy	Precision	Recall	F-score
1	2011	Malimg	Nataraj et al. (2011)	K-NN	98.08%	–	–	–
2	2017	Malimg	Luo and Lo (2017)	CNN+LBP	93.72%	94.13%	92.54%	93.33%
3	2017	Malimg	Rezende et al. (2017)	ResNet50	97.48%	–	–	–
4	2017	Malimg	Rezende et al. (2017)	ResNet-50	98.62%	–	–	–
5	2017	Malimg	Makandar and Patrot (2017)	GIST+SVM	98.88%	–	–	–
6	2018	Malimg	Lo et al. (2019)	Xception	98.52%	–	–	–
7	2019	Malimg	Bhodia et al. (2019)	ResNet34	94.80%	–	–	–
8	2019	Malimg	Roseline et al. (2019)	Ensembling using RF	97.82%	98%	98%	98%
9	2019	Malimg	Agarap (2017)	CNN-SVM	77.22%	84%	77%	79%
10	2019	Malimg	Ben Abdel Ouahab et al. (2019)	K-NN	97%	–	–	–
11	2019	Malimg	Gibert et al. (2019)	CNN	97.18%	–	–	–
12	2019	Malimg	Vinayakumar et al. (2019)	CNN+LSTM	96.3%	96.3%	96.2%	96.2%
13	2019	Malimg	Vinayakumar et al. (2019)	RF	78.6%	–	–	–
14	2019	Malimg	Vinayakumar et al. (2019)	NB	80.5%	–	–	–
15	2019	Malimg	Vinayakumar et al. (2019)	KNN	41.8%	–	–	–
16	2019	Malimg	Vinayakumar et al. (2019)	DT	79.5%	–	–	–
17	2019	Malimg	Singh et al. (2019)	ResNet50	96.08%	95.76%	96.16%	95.96%
18	2020	Malimg	Vasan et al. (2020b)	VGG16,	97.59%	–	–	–
19	2020	Malimg	Vasan et al. (2020b)	ResNet50	95.94%	–	–	–
20	2021	Malimg	Hemalatha et al. (2021)	DenseNet	98.23%	97.78%	97.92%	97.85%
21	**2022**	**Malimg**	**Proposed framework**		**99.06%**	**98.47%**	**98.52%**	**98.49%**

the existing works. To validate the detection effectiveness of our proposed framework, we conducted experiments with the Malimg dataset. Fig. 5 illustrates the comparative results of the proposed framework with state-of-the-art to have better visualisation and understanding.

In our proposed framework, RegNetY320 with SVM has outperformed. We used the same combination for comparison with the state-of-the-art. It is palpable that the proposed framework has outperformed the existing work on a benchmark dataset as well. We are unaware that any previous work has handled the problem of an imbalanced dataset and extensive malware analysis in single research work.

4.4. Performance analysis on a smaller dataset

This section validates the detection effectiveness of the proposed framework on a smaller dataset. DatasetD is used to conduct experiments for this sub-section. DatasetD consists of 1000 malicious PEs and 1000 benign PEs. These PEs are randomly selected from datasetA. The dataset is split into training, validation, and testing with a ratio of 60:20:20. 600 malicious PEs and 600 benign PEs are used for training, and 200 of each class are used for validation and testing. Deep learning models are called data-hungry models. They give better results

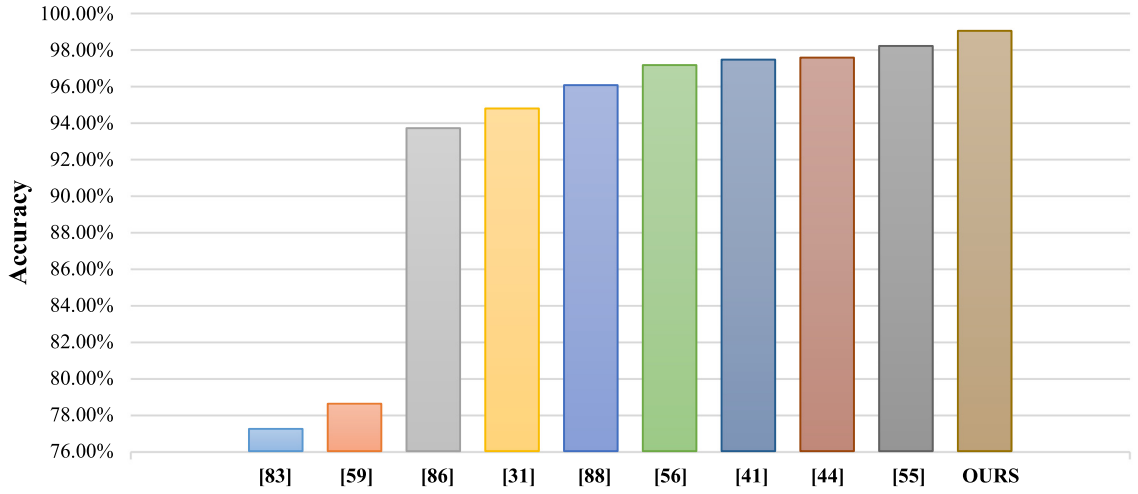


Fig. 5. A graphical comparison of the proposed framework with state-of-the-art.

when trained with a large amount of data. Due to privacy issues, the latest attack patterns are unavailable in real life. The results are not reasonable if the learning-based malware detectors are trained with an imbalanced class instance. The deep learning models tend to skew towards the majority class and cannot generalise the instances from the minority class well. The performance analysis of imbalanced datasets and its solution has been discussed in the previous sections. In this section, we validate the performance of the proposed framework on a limited-size, smaller dataset (with few samples in training). The performance of the proposed framework is validated on different settings of hyperparameters, i.e., the number of epochs and batch sizes with 15 fine-tuned deep learning-based and CNN-SVM malware detectors. We also conducted experiments with different values of batch size, including 8, 16, and 32, with 150 epochs. The performance of the state-of-the-art fine-tuned malware detectors decreased with a smaller dataset.

However, the proposed malware detector outperformed with an accuracy of 93.43%. In our previous experiments with datasetA, datasetB, and datasetC, the VGG16 performed better than the other deep learning models. The VGG16 has 89.71% accuracy with 150 epochs and 32 batch sizes. The CNN-SVM has increased accuracy to 90.87%, which is 1.16% better than the VGG16 (see Table 16).

The proposed framework used RegNetY320 as a feature extractor. The extracted deep learning features are then given to SVM for final malware detection. The proposed framework gave 93.43% accuracy. The proposed framework has outperformed VGG16, VGG19, and CNN-SVM with an accuracy of 3.72%, 4.7%, and 2.56%, respectively.

4.5. Performance comparison of detectors: With one vs. all features

This section presents the analysis of the one-vs-all features image-based dataset. Al-Dujaili et al. in Al-Dujaili et al. (2018) proposed a method of malware detection using API features. Each PE file is converted into a corresponding feature vector. The authors then used an artificial neural network (ANN) for further malware detection. Our work presents a hybrid deep learning-based solution for malware detection. We conducted experiments with four perspectives. Table 17 presents these results. The first data row shows the detection effectiveness of Al-Dujaili et al. (2018). The subsequent rows demonstrate the results of our proposed approaches. The experiments were performed using the same dataset as Al-Dujaili et al. (2018).

In *Proposed Method 1* and *Proposed Method 2,* we converted API-based feature vectors into images and named the dataset as APIImg. The process mentioned in Section 3.1.2 transformed all the feature vectors into images.

In *Proposed Method 1*, we have used fine-tuned VGG16 model. Out of 15 fine-tuned models, VGG16 outperformed other models once applied end-to-end for malware detection. We then leverage the experiments and apply fine-tuned VGG16 to compare with the performance of ANN used by Al-Dujaili et al. (2018). The *Proposed Method 1* classified the malware and benign samples with an accuracy of 93.47% and outperformed with 1.57% accuracy.

In *Proposed Method 2*, we have conducted the experiments using our proposed framework. We have extracted the features from the APIImg dataset using RegNetY320. The deep features are then fed to SVM for final detection. The *Proposed Method 2* gave an accuracy of 97.86%. The accuracy of the proposed framework is 5.96% and 4.39% better than (Al-Dujaili et al., 2018) and fine-tuned VGG16, respectively.

In *Proposed Method 3* and *Proposed Method 4*, we have transformed all the binaries of the source PEs dataset (Al-Dujaili et al., 2018) into images. The fine-tuned VGG16 gave 94.02% accuracy, and the proposed framework outperformed with 98.53%.

Through these experiments, we analysed the detection effectiveness of whether to consider a single feature or full binaries for malware detection. In Al-Dujaili et al. (2018), the proposed method worked on the binary feature vectors with an ANN for detection. We have analysed the performance that converting those binary feature vectors into images and leveraging the latest advancement in deep learning can certainly increase the accuracy. We concluded that applying the latest deep learning models could improve accuracy. In method 3 and method 4, we analysed that considering 1 feature (e.g., API calls) regardless of what would be the next detection model or method (binary feature vectors or visualisation) could reduce the accuracy. We have transformed the PEs binaries into images and applied the latest development in deep learning to analyse the performance. The applied models have certainly increased detection effectiveness. We are unsure how cybercriminals manipulate the PEs to convert them into malicious ones. Thus, considering all the binaries proved helpful in increasing the detection accuracy (method 3 and method 4).

4.6. Statistical results

Deep learning models can be employed for feature computation and final classification. We have conducted experiments from both perspectives. Firstly, we leverage the power of deep learning models for end-to-end malware detection. Secondly, deep features are extracted using deep learning models. The deep learning models efficiently extract deep features by reducing the overhead of feature engineering and the need for domain experts. VGG16 has performed better for end-to-end malware detection than other fine-tuned deep learning models. In the

Table 16
A comparison of the proposed framework on a smaller dataset.

Sr. No.	Model	Epochs	Batch size	Valid loss	Valid Acc.	Test loss	Test Acc.
1	VGG16	150	8	0.8283	84.15%	0.5161	88.19%
		150	16	0.8176	84.07%	0.4513	88.76%
		150	32	0.6356	84.40%	0.3665	89.71%
		250	32	0.7702	84.28%	0.418	89.51%
		150	64	0.8612	83.64%	0.3687	89.76%
		250	64	0.6013	86.10%	0.3585	**89.84%**
2	ResNet50	150	8	43.87	42.77%	45.75	42.44%
		70	8	47.27	42.86%	47.1	42.95%
		150	16	9.015	39.65%	10.79	39.91%
		150	32	49.6	42.90%	49.44	**43.01%**
3	InceptionV3	150	8	4.855	66.70%	3.79	69.82%
		150	16	3.5	63.31%	2.462	**70.04%**
		250	16	4.04	64.37%	3.23	68.89%
		150	32	2.917	63.10%	2.391	66.60%
4	VGG19	150	8	0.8895	84.19%	0.3683	**89.34%**
		150	16	0.774	85.93%	0.3878	89.44%
		150	32	0.5388	85.46%	0.3769	88.73%
5	MobileNet	150	8	15.59	60.64%	11.92	63.70%
		150	16	12.67	61.50%	7.18	**67.10%**
		250	16	16.43	60.02%	11.41	62.14%
		150	32	18.96	53.09%	12.55	54.92%
6	DenseNet169	150	8	6.65	72.95%	5.57	**75.77%**
		150	16	5.92	72.61%	5.03	73.04%
		150	32	3.82	73.37%	3.59	75.47%
		250	32	3.94	72.95%	4.01	73.65%
7	Xception	150	8	7.6	60.19%	8.69	**56.27%**
		150	16	7.23	58.75%	8.42	56.00%
		150	32	5.42	59.13%	7.46	53.14%
8	DenseNet201	150	8	19.16	58.20%	21.06	59.31%
		150	16	12.17	61.29%	12.2	61.64%
		150	32	8.28	63.86%	8.26	64.00%
		150	64	5.93	61.96%	5.15	66.23%
		250	64	7.44	62.26%	7.9	63.70%
		250	32	8.44	64.03%	6.99	**68.32%**
9	Inception ResNet V2	150	8	6.3	60.78%	7.4	**56.88%**
		150	16	6.92	55.62%	9.28	51.58%
		150	32	5.54	56.13%	7.19	51.95%
10	MobileNetV2	150	8	17.03	54.31%	16.03	52.97%
		150	16	11.35	53.47%	11.26	**54.18%**
		150	32	14.84	49.66%	16.22	49.82%
11	ResNet152V2	150	8	15.41	66.23%	14.06	67.74%
		150	16	11.85	67.54%	9.69	**70.24%**
		150	32	7.874	68.30%	7.15	69.50%
12	NasNetMobile	150	8	6.43	62.55%	5.44	64.34%
		150	16	2.95	68.34%	3.24	67.64%
		150	32	2.2	68.05%	1.84	**69.57%**
		250	32	3.19	64.88%	2.67	66.83%
		150	64	2.41	61.71%	2.11	62.21%
		250	64	2.37	64.29%	1.73	69.33%
13	AlexNet	150	8	13.98	64.47%	11.49	63.32%
		150	16	13.64	56.13%	13.08	**64.21%**
		150	32	11.09	59.75%	10.96	62.98%
14	SqueezeNet	150	8	13.75	57.41%	13.38	58.29%
		150	16	14.51	61.44%	11.27	62.55%
		150	32	15.48	67.79%	12.08	**64.47%**
15	RegNetY320	150	8	6.83	71.17%	6.72	72.74%
		150	16	6.08	72.86%	6.95	**74.08%**
		150	32	5.72	73.58%	5.67	73.18%
16	CNN-SVM	150	8	0.7415	89.16%	0.3741	89.93%
		150	16	0.8749	89.27%	0.3686	89.87%
		150	32	0.4479	89.38%	0.3277	**90.87%**
17	Proposed detector	150	32	0.2177	91.85%	0.2974	**93.43%**

Table 17
A comparison of one-vs-all features image dataset (all metrics are in %, except epochs column).

Sr. No.	Methods	Dataset	Model	Epochs	Accuracy	Specificity	Sensitivity	FPR	FNR
1	Al-Dujaili et al. (2018)	VirusShare-API	ANN	150	91.9	91.8	91.9	8.2	8.1
2	Proposed Method 1	VirusShare-APIImg	VGG16	150	93.47	93.08	93.87	6.92	6.13
3	Proposed Method 2	VirusShare-APIImg	Proposed framework	150	97.86	97.82	97.9	2.18	2.1
4	Proposed Method 3	VirusShare	VGG16	150	94.02	94.23	93.81	5.77	6.19
5	Proposed Method 4	VirusShare	Proposed framework	150	98.53	98.74	98.33	1.26	1.67

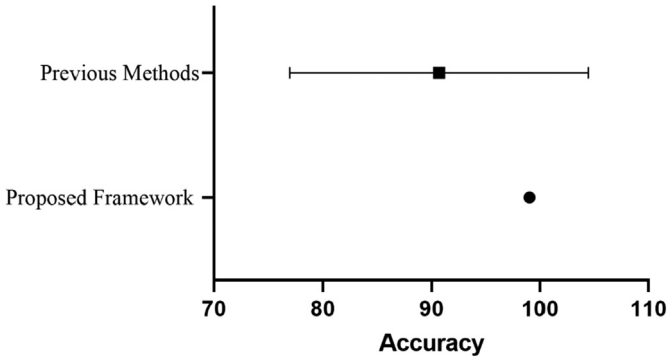


Fig. 6. Wilcoxon signed-rank test.

second scenario, integrating RegNetY320 (as a feature extractor) with SVM outperformed other deep learning models. VGG16 is not considered a state-of-the-art benchmark model for the ImageNet dataset. The latest deep learning models have outperformed VGG16 on the ImageNet dataset. In this research, the experiments are performed with different perspectives using the latest state-of-the-art deep learning models both for feature extraction and end-to-end malware detection. The results revealed the deterioration of performance with the latest models. These results are interesting in the context of malware detection, as the latest deep learning models outperformed VGG16 on the ImageNet dataset. We inferred from the results that the learned features of deep learning models, except VGG16, are less transferable. It is hard to give the exact rationale behind what actually makes the CNN models more transferable to each other; however, the results show that one CNN model might outperform other state-of-the-art models on a specific task. It would be valuable to evaluate the transformability of a CNN model for a specific task in future work. However, performing experiments with various aspects confirmed the superiority of our proposed malware detection framework. We have conducted a non-parametric Wilcoxon signed-rank (WSR) test to show the statistical significance of our proposed framework.

The null hypothesis statistical testing (NHST) is a statistical technique to interpret the results. It also ensures that the claim of better performance is backed up by statistical analysis. The WSR test is used to compare the location of two populations using the same dataset or to locate the population based on sample data. The WRS test is the best alternative to the t-test when the population means are not of interest. For example, when we want to test whether there are >50% chances that a sample from one population is better than the samples from other populations or whether the median value of the population is non-zero. Secondly, the existing methods mentioned in Table 15 do not follow the Gaussian distribution. Thus, we have chosen one sample non-parametric WSR test to validate the significance of theoretical and median values from the existing methods. The whiskers in Fig. 6 show the standard division of different methods.

The null hypothesis (H_0) is that our proposed malware detection framework does not have better detection effectiveness than existing methods. The alternative hypothesis (H_a) is that the proposed malware detection framework has better detection effectiveness and is performed differently. The test is performed to validate the $P \ll 0.0001$ value using Table 15. 10-fold cross-validation is further used to avoid any biased effect of the data. The test validates whether the existing methods are significantly the same or different and whether our proposed framework significantly improved the detection effectiveness. The two-tailed p-value is less than the threshold α value (0.0001). This shows that the improvement of the proposed framework is highly statistically significant and rejects the null hypothesis. The statistical test rejected the null hypothesis based on the results reported in Table 15. Our proposed framework has significantly improved the effectiveness of malware detection in various aspects. Table 18 summarises the comparison of the existing system and the proposed approach based on various indicators. The existing systems include static analyses, dynamic analyses, and existing learning-based solutions.

Time Complexity The time complexity determines the time a model will take during training and testing. The time complexity and time are directly proportional. The more complexity, the great deal of time the model will take. Deep learning-based malware detectors contain complex structures; thus, evaluating a model or making a quick prediction about a PE file is hard. The time complexity of a convolutional layer depends upon the size of kernels, the size of input feature maps, and the count of input and output channels of an image. Given D as the depth of the CNN model, S_m as the sizes of feature maps, C_{in} and C_{out} as the count of input channels and output channels, n as the name of a convolutional layer, S_K as the sizes of kernels, the Eq. (13) represents the time complexity of a CNN model:

$$O\left(\sum_{n=1}^{D} S_m^2 * S_k^2 * C_{n-1} * C_n\right) \tag{13}$$

Limitations: The proposed framework also comes with a few limitations. The dimensions of the features extracted using deep learning models are higher. We need a significant amount of memory to hold those 1D features temporarily. In future, we will explore the implementation of incremental learning. In incremental learning, only an incremental subset of data is needed for training. Exploring various feature reduction and selection techniques is another open research direction. The malware and benign images are translated into images. These transformations act on the image data, but how those transformations translate back onto the portable executable is unclear. This means it is possible that even a small change to the malware

Table 18
A comparative summary of existing approaches with the proposed approach.

	Existing approaches	Proposed approach
Method	Learning-based solutions use deep learning for end-to-end classification.	Use deep learning for feature extraction and machine learning as the final detector.
Efficient	Require a lot of training for end-to-end classification, extensive resources and slower in the case of dynamic analysis.	Efficient as it uses a deep learning model for feature extraction and eliminates the need for the virtual environment.
Flexibility	Can work with one model. Most existing learning-based solutions use and experiment with one deep learning model.	Experimented with 15 deep learning and 12 machine learning models. Flexible to adapt any deep learning and/or machine learning model.
Adaptive	Need specific approaches for each field (IoT, mobile malware).	Adaptive to IoT and attacks on mobile devices and easily translate them into images.
Decoding	Need to dissemble and decode the file for static analysis.	Easily mapped the whole file contents into binary and images later on.
Domain knowledge Reverse engineering	Need domain knowledge and experts. Yes	No need for domain experts. No
Memory resources	Does not require extensive memory resources as the features are decoded into other formats.	Yes, to hold the extracted features using a deep learning model.
Extensive virtual environment	Require extensive resources in the case of dynamic analysis.	Does not require any virtual environment or extensive resources.
Features count	Most of the time, consider one feature for detection, e.g., API or Opcode.	Considers all file for file binaries into an image for further processing.
Colour format	Mostly grey images in the case of learning-based solutions.	RGB Colour images to have more rich features for malware detection.
Imbalanced data	Not handled	Handled through data augmentation

image (e.g. rotating it 2 degrees) could represent a potentially benign portable executable. It would be optimal to either analyse what portable executable's images result in.

Future Works: We will explore the robustness of the proposed framework against adversarial attacks. Due to the breadth and depth of analysis, other important metrics (such as the area under the curve and the F-measure) are overlooked in analysing the proposed framework. We will analyse the performance of the proposed framework against these metrics in future. We will analyse the effectiveness of the proposed framework for Android and IoT applications. Deep learning models are CNN-based, which comes with a limitation to taking uniform-size images for training. We will explore a spatial pyramid pooling layer in the future that can take input of any size. This will enable us to utilise all the images in an original format without normalisation. Having stated the limitations, it is evident from the results that the proposed framework can be useful in real-world applications in detecting malicious PEs. The proposed approach is generic, flexible, and adaptive to other deep learning and machine learning models. Any new deep learning and machine learning model can be investigated and integrated. In the defence industry, the proposed framework will be helpful in devising more efficient malware detection solutions.

5. Conclusions

This paper has proposed a framework for malware detection based on a hybrid deep learning and machine learning approach, as well as providing an in-depth analysis of various methods for malware detection. The proposed method combined deep learning with machine learning and did not need intensive feature engineering and domain knowledge. It consisted of three major steps: visualising a portable executable file as a colour image, extracting deep features from the colour image using a fine-tuned deep learning model, and detecting malware based on the deep features using support vector machines.

The performance of the proposed framework has been validated on 15 deep learning and 12 machine learning models. The proposed framework is scalable, cost-effective, and efficient. Any deep learning and machine model can be used as a feature extractor and final detector, respectively. We found the best combination with RegNetY320 as the feature extractor and SVM as the final detector. The combination outperformed existing methods with an accuracy of 99.06%. Resultantly, we eliminated the need for knowledge from domain experts for reverse engineering tasks. Furthermore, the proposed framework uses deep learning, thus eliminating the high computation and time-consuming overhead of feature engineering. Lastly, the success of a learning-based detector highly depends upon the features on which the model was trained. The paper provides the detection effectiveness of various models on the dataset generated using one or all features for malware detection. Extensive experiments have demonstrated the superiority of our proposed framework over state-of-the-art approaches.

The statistical test was performed to validate the hypothesis that the proposed model generalises better. The validity of the proposed framework was validated over various learning-based approaches. The results demonstrate convincing statistical evidence that the proposed framework performed better than other state-of-the-art approaches.

There is a need for more advanced malware visualisation techniques in the future. It was apparent that reshaping and resizing images influenced the detection effectiveness of the learning-based models. We will explore designing visualisation techniques specifically for PEs transformation (When dealing with real-world images, bicubic interpolation is usually used). We will also analyse the effect of other data resampling methods, such as Generative adversarial network (GAN), on the learning process. The proposed framework is applicable to all Windows PEs. However, in the future, we will explore the performance of the proposed framework on other Mobile and IoT datasets. We will also explore whether the feature reduction and selection techniques help to improve detection accuracy. Lastly, we will explore the impact of ensemble techniques as a final detector.

CRediT authorship contribution statement

Kamran Shaukat: Conceptualization, Data curation, Formal analysis, Investigation, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Suhuai Luo:** Conceptualization, Investigation, Methodology, Project administration, Resources, Validation, Writing – review & editing. **Vijay Varadharajan:**

Conceptualization, Investigation, Methodology, Project administration, Resources, Validation, Writing – review & editing.

Declaration of competing interest

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

Data availability

Data will be made available on request.

References

Agarap, A.F., 2017. Towards building an intelligent anti-malware system: a deep learning approach using support vector machine (SVM) for malware classification. arXiv preprint arXiv:1801.00318.

Agrawal, R., Srikant, R., 1995. Mining sequential patterns. In: Proceedings of the Eleventh International Conference on Data Engineering. IEEE, pp. 3–14.

Akram, Z., Majid, M., Habib, S., 2021. A systematic literature review: Usage of logistic regression for malware detection. In: 2021 International Conference on Innovative Computing. ICIC, IEEE, pp. 1–8.

Al-Dujaili, A., Huang, A., Hemberg, E., O'Reilly, U.-M., 2018. Adversarial deep learning for robust detection of binary encoded malware. In: 2018 IEEE Security and Privacy Workshops. SPW, IEEE, pp. 76–82.

Al-Hashmi, A.A., et al., 2022. Deep-ensemble and multifaceted behavioral malware variant detection model. IEEE Access 10, 42762–42777.

Anderson, B., Quist, D., Neil, J., Storlie, C., Lane, T., 2011. Graph-based malware detection using dynamic analysis. J. Comput. Virol. 7 (4), 247–258.

Arora, A., Garg, S., Peddoju, S.K., 2014. Malware detection using network traffic analysis in android based mobile devices. In: 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies. IEEE, pp. 66–71.

Awan, M.J., et al., 2021. Image-based malware classification using VGG19 network and spatial convolutional attention. Electronics 10 (19), 2444.

Bansal, M., Kumar, M., Sachdeva, M., Mittal, A., 2021. Transfer learning for image classification using VGG19: Caltech-101 image data set. J. Ambient Intell. Humaniz. Comput. 1–12.

Ben Abdel Ouahab, I., Bouhorma, M., Boudhir, A.A., El Aachak, L., 2019. Classification of grayscale malware images using the K-nearest neighbor algorithm. In: The Proceedings of the Third International Conference on Smart City Applications. Springer, pp. 1038–1050.

Bhodia, N., Prajapati, P., Di Troia, F., Stamp, M., 2019. Transfer learning for image-based malware classification. arXiv preprint arXiv:1903.11551.

Bouchaib, P., Bouhorma, M., 2021. Transfer learning and smote algorithm for image-based malware classification. In: Proceedings of the 4th International Conference on Networking, Information Systems & Security. pp. 1–6.

Cesare, S., Xiang, Y., Zhou, W., 2013. Control flow-based malware variantdetection. IEEE Trans. Dependable Secure Comput. 11 (4), 307–317.

Chandio, A., et al., 2022. Precise single-stage detector. arXiv preprint arXiv:2210.04252.

Chen, L., 2018. Deep transfer learning for static malware classification. arXiv preprint arXiv:1812.07606.

Cogswell, M., Ahmed, F., Girshick, R., Zitnick, L., Batra, D., 2015. Reducing overfitting in deep networks by decorrelating representations. arXiv preprint arXiv:1511.06068.

Cortes, C., Vapnik, V., 1995. Support-vector networks. Mach. Learn. 20 (3), 273–297.

Cui, Z., Du, L., Wang, P., Cai, X., Zhang, W., 2019. Malicious code detection based on CNNs and multi-objective algorithm. J. Parallel Distrib. Comput. 129, 50–58.

Cunningham, P., Delany, S.J., 2021. K-nearest neighbour classifiers-a tutorial. ACM Comput. Surv. 54 (6), 1–25.

D'Angelo, G., Ficco, M., Palmieri, F., 2020. Malware detection in mobile environments based on autoencoders and API-images. J. Parallel Distrib. Comput. 137, 26–33.

De Paola, A., Gaglio, S., Re, G.L., Morana, M., 2018. A hybrid system for malware detection on big data. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, pp. 45–50.

El-Shafai, W., Almomani, I., AlKhayer, A., 2021. Visualized malware multi-classification framework using fine-tuned CNN-based transfer learning models. Appl. Sci. 11 (14), 6446.

Frank, E., Hall, M.A., 2011. Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann.

Fu, Z., Ding, Y., Godfrey, M., 2021. An LSTM-based malware detection using transfer learning. J. Cybersecur. 3 (1), 11.

Fujino, A., Murakami, J., Mori, T., 2015. Discovering similar malware samples using api call topics. In: 2015 12th Annual IEEE Consumer Communications and Networking Conference. CCNC, IEEE, pp. 140–147.

Galen, C., Steele, R., 2021. Empirical measurement of performance maintenance of gradient boosted decision tree models for malware detection. In: 2021 International Conference on Artificial Intelligence in Information and Communication. ICAIIC, IEEE, pp. 193–198.

Gao, X., Hu, C., Shan, C., Liu, B., Niu, Z., Xie, H., 2020. Malware classification for the cloud via semi-supervised transfer learning. J. Inf. Secur. Appl. 55, 102661.

Gibert, D., Mateu, C., Planes, J., Vicens, R., 2019. Using convolutional neural networks for classification of malware represented as images. J. Comput. Virol. Hacking Tech. 15 (1), 15–28.

Gibert, D., Planes, J., Mateu, C., Le, Q., 2022. Fusing feature engineering and deep learning: A case study for malware classification. Expert Syst. Appl. 117957.

Guo, H., Cheng, H.K., Kelley, K., 2016. Impact of network structure on malware propagation: A growth curve perspective. J. Manage. Inf. Syst. 33 (1), 296–325.

Han, K., Lim, J.H., Im, E.G., 2013. Malware analysis method using visualization of binary files. In: Proceedings of the 2013 Research in Adaptive and Convergent Systems. pp. 317–321.

Hemalatha, J., Roseline, S.A., Geetha, S., Kadry, S., Damaševičius, R., 2021. An efficient densenet-based deep learning model for malware detection. Entropy 23 (3), 344.

Huda, S., Abawajy, J., Alazab, M., Abdollalihian, M., Islam, R., Yearwood, J., 2016. Hybrids of support vector machine wrapper and filter based framework for malware detection. Future Gener. Comput. Syst. 55, 376–390.

Huo, D., Li, X., Li, L., Gao, Y., Li, X., Yuan, J., 2022. The application of 1D-CNN in microsoft malware detection. In: 2022 7th International Conference on Big Data Analytics. ICBDA, IEEE, pp. 181–187.

Imran, M., Afzal, M.T., Qadir, M.A., 2015. Using hidden markov model for dynamic malware analysis: First impressions. In: 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery. FSKD, IEEE, pp. 816–821.

Jiang, B., Chen, S., Wang, B., Luo, B., 2022. MGLNN: Semi-supervised learning via multiple graph cooperative learning neural networks. Neural Netw. 153, 204–214.

Jiang, Y., Li, R., Tang, J., Davanian, A., Yin, H., 2020. Aomdroid: Detecting obfuscation variants of android malware using transfer learning. In: International Conference on Security and Privacy in Communication Systems. Springer, pp. 242–253.

Kadri, M.A., Nassar, M., Safa, H., 2019. Transfer learning for malware multi-classification. In: Proceedings of the 23rd International Database Applications & Engineering Symposium. pp. 1–7.

Kim, S., 2018. PE header analysis for malware detection.

Kolter, J.Z., Maloof, M.A., 2006. Learning to detect and classify malicious executables in the wild. J. Mach. Learn. Res. 7 (12).

Kumar, S., 2021. MCFT-CNN: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in internet of things. Future Gener. Comput. Syst. 125, 334–351.

Kumar, R., Subbiah, G., 2022. Zero-day malware detection and effective malware analysis using Shapley ensemble boosting and bagging approach. Sensors 22 (7), 2798.

Lad, S.S., Adamuthe, A.C., 2020. Malware classification with improved convolutional neural network model. Int. J. Comput. Netw. Inf. Secur. 12, 30–43.

Li, N., Zhang, Z., Che, X., Guo, Z., Cai, J., 2021. A survey on feature extraction methods of heuristic malware detection. J. Phys. Conf. Ser. 1757 (1), 012071, IOP Publishing.

2021. LIEF - library to instrument executable formats - quarkslab. https://lief.quarkslab.com/ (accessed April 05, 2021).

Lo, W.W., Yang, X., Wang, Y., 2019. An xception convolutional neural network for malware classification with transfer learning. In: 2019 10th IFIP International Conference on New Technologies, Mobility and Security. NTMS, IEEE, pp. 1–5.

Luo, J.-S., Lo, D.C.-T., 2017. Binary malware image classification using machine learning with local binary pattern. In: 2017 IEEE International Conference on Big Data (Big Data). IEEE, pp. 4664–4667.

Makandar, A., Patrot, A., 2017. Malware class recognition using image processing techniques. In: 2017 International Conference on Data Management, Analytics and Innovation. ICDMAI, IEEE, pp. 76–80.

Marastoni, N., Giacobazzi, R., Dalla Preda, M., 2021. Data augmentation and transfer learning to classify malware images in a deep learning context. J. Comput. Virol. Hacking Tech. 17 (4), 279–297.

Martín, A., Lara-Cabrera, R., Camacho, D., 2019. Android malware detection through hybrid features fusion and ensemble classifiers: the AndroPyTool framework and the OmniDroid dataset. Inf. Fusion 52, 128–142.

Maulana, P., Heryanto, A., Oklilas, A.F., 2022. Klasifikasi Malware Adware Pada Android Menggunakan Metode Support Vektor Machine (SVM) Dan Linear Discriminant Analysis. LDA, Sriwijaya University.

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

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

Nahmias, D., Cohen, A., Nissim, N., Elovici, Y., 2020. Deep feature transfer learning for trusted and automated malware signature generation in private cloud environments. Neural Netw. 124, 243–257.

Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S., 2011. Malware images: visualization and automatic classification. In: Proceedings of the 8th International Symposium on Visualization for Cyber Security. pp. 1–7.

Oliva, A., Torralba, A., 2001. Modeling the shape of the scene: A holistic representation of the spatial envelope. Int. J. Comput. Vis. 42 (3), 145–175.

Prajapati, P., Stamp, M., 2021. An empirical analysis of image-based learning techniques for malware classification. In: Malware Analysis using Artificial Intelligence and Deep Learning. Springer, pp. 411–435.

Prima, B., Bouhorma, M., 2020. Using transfer learning for malware classification. Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci. 44, 343–349.

Radosavovic, I., Kosaraju, R.P., Girshick, R., He, K., Dollár, P., 2020. Designing network design spaces. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10428–10436.

Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.K., 2018. Malware detection by eating a whole exe. In: Workshops At the Thirty-Second AAAI Conference on Artificial Intelligence.

Rezende, E., Ruppert, G., Carvalho, T., Ramos, F., De Geus, P., 2017. Malicious software classification using transfer learning of resnet-50 deep neural network. In: 2017 16th IEEE International Conference on Machine Learning and Applications. ICMLA, IEEE, pp. 1011–1014.

Rong, C., Gou, G., Cui, M., Xiong, G., Li, Z., Guo, L., 2020. TransNet: Unseen malware variants detection using deep transfer learning. In: International Conference on Security and Privacy in Communication Systems. Springer, pp. 84–101.

Roseline, S.A., Sasisri, A., Geetha, S., Balasubramanian, C., 2019. Towards efficient malware detection and classification using multilayered random forest ensemble technique. In: 2019 International Carnahan Conference on Security Technology. ICCST, IEEE, pp. 1–6.

Rosenberg, I., Sicard, G., David, E.O., 2018. End-to-end deep neural networks and transfer learning for automatic analysis of nation-state malware. Entropy 20 (5), 390.

Ross, Q.J., 1993. C4. 5: Programs for Machine Learning. San Mateo, CA.

Roy, A.M., Bhaduri, J., Kumar, T., Raj, K., 2022. WilDect-YOLO: An efficient and robust computer vision-based accurate object localization model for automated endangered wildlife detection. Ecol. Inform. 101919.

Schölkopf, B., Williamson, R.C., Smola, A.J., Shawe-Taylor, J., Platt, J.C., 2000. Support vector method for novelty detection. Adv. Neural Inf. Process. Syst. 582–588.

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

Searles, R., et al., 2017. Parallelization of machine learning applied to call graphs of binaries for malware detection. In: 2017 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing. PDP, IEEE, pp. 69–77.

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

Shaid, S.Z.M., Maarof, M.A., 2014. Malware behaviour visualization. J. Teknol. 70 (5).

Shaukat, K., Luo, S., Chen, S., Liu, D., 2020a. Cyber threat detection using machine learning techniques: A performance evaluation perspective. In: 2020 International Conference on Cyber Warfare and Security. ICCWS, IEEE, pp. 1–6.

Shaukat, K., Luo, S., Varadharajan, V., 2022. A novel method for improving the robustness of deep learning-based malware detectors against adversarial attacks. Eng. Appl. Artif. Intell. 116, 105461.

Shaukat, K., Luo, S., Varadharajan, V., Hameed, I.A., Xu, M., 2020b. A survey on machine learning techniques for cyber security in the last decade. IEEE Access 8, 222310–222354.

Shaukat, K., et al., 2020c. Performance comparison and current challenges of using machine learning techniques in cybersecurity. Energies 13 (10), 2509.

Singh, A., Handa, A., Kumar, N., Shukla, S.K., 2019. Malware classification using image representation. In: International Symposium on Cyber Security Cryptography and Machine Learning. Springer, pp. 75–92.

Tang, Y., 2013. Deep learning using linear support vector machines. arXiv preprint arXiv:1306.0239.

Vasan, D., Alazab, M., Wassan, S., Naeem, H., Safaei, B., Zheng, Q., 2020a. IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. Comput. Netw. 171, 107138.

Vasan, D., Alazab, M., Wassan, S., Safaei, B., Zheng, Q., 2020b. Image-based malware classification using ensemble of CNN architectures (IMCEC). Comput. Secur. 92, 101748.

Vinayakumar, R., Alazab, M., Soman, K., Poornachandran, P., Venkatraman, S., 2019. Robust intelligent malware detection using deep learning. IEEE Access 7, 46717–46738.

VirusShare, https://virusshare.com/ (accessed May 24, 2022).

Wang, W., Zhao, M., Wang, J., 2019. Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. J. Ambient Intell. Humaniz. Comput. 10 (8), 3035–3043.

Zhao, Y., Cui, W., Geng, S., Bo, B., Feng, Y., Zhang, W., 2020. A malware detection method of code texture visualization based on an improved faster RCNN combining transfer learning. IEEE Access 8, 166630–166641.